Premios para los lectores • Estrategia • C y Ensamblador • Aventura • DIV 2 • Arcade

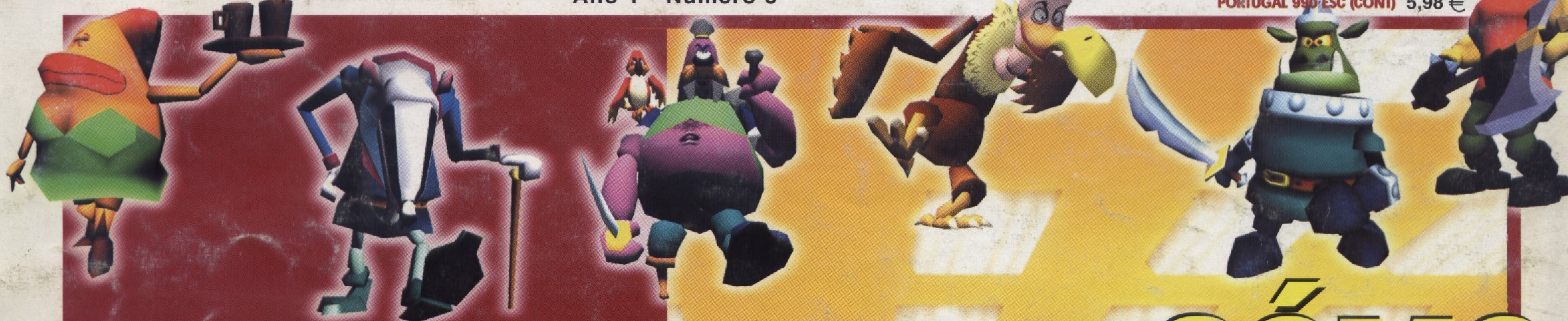


CONCURSO JUEGOS DIV

DIV manía

995 ptas.
PORTUGAL 990 ESC (CONT) 5,98 €

www.prensatecnica.com
Año 1 • Número 5



CÓMO CREAR UN VIDEOJUEGO

• Especial programación
Los 10 mandamientos del programador independiente

•Iniciación DIV
Los entresijos de nuestro programa favorito

•Direct X
De la teoría pasamos a la práctica

•Webs DIV
Grupos de desarrollo con proyectos calientes

•Programación
Programa juegos de tu género preferido

•Diseño y dibujo
Creamos la presentación de un videojuego

•Photoshop
Trabajando con capas y canales

Prens@ Técnica


00005
8 413042 087533

Todo para archivar tus CDs

numain



CDs vírgenes
ref.: CD-R74 (74 min.)
ref.: CD-R80 (80 min.)
ref.: CD-RW (regrabable)



Estuche Ref.:CD24
Capacidad para 24 CDs



Estuche Ref.:CD48
Capacidad para 48 CDs



Archivo Ref.:ARCH60
Capacidad para 60 CDs



Estuche Ref.:CD80
Capacidad para 80 CDs



Estuche Ref.:CD160
Capacidad para 160 CDs



Archivo Ref.:CLST80
Capacidad para 80 CDs



Estuche Ref.:ESC48
Capacidad para 48 CDs



Estuche Ref.:ESC24
Capacidad para 24 CDs



Torre diseño Ref.:SST-50
Capacidad para 50 CDs



Torre diseño Ref.:SST-30
Capacidad para 30 CDs



Torre diseño Ref.:SST-20
Capacidad para 20 CDs



Estuche Ref.:CD-50P
Capacidad para Discman
+ 6 CDs con caja



Estuche Ref.:CD-15
Capacidad para 15 CDs con caja



Estuche Ref.:CD-240P
Capacidad para Discman + 24 CDs



Regalo seguro

Al realizar un pedido superior a 12.000 ptas

Ref.:CD-20R
Base de sobremesa,
capacidad para 20 CDs



Ref.:NBC Maletin ordenador portátil



Ref.:CD-60T/W
Capacidad para
60 CDs



Ref.:CD-120T/W
Capacidad para
120 CDs



Ref.:CD-80T/W
Capacidad
para 80 CDs



Ref.:CD-160T/W
Capacidad para
160 CDs

Pidenos catálogo o haz tu pedido por teléfono, Fax,
E-mail indicando las referencias correspondientes

Gastos de envío gratuitos para pedidos superiores a 12.000 ptas.

- ❑ ref. CD-R74 *(145 ptas.)*
- ❑ ref. CD-R80 *(195 ptas.)*
- ❑ ref. CD-RW *(350 ptas.)*
- ❑ ref. CD24 *(990 ptas.)*
- ❑ ref. CD48 *(1.500 ptas.)*
- ❑ ref. CD80 *(2.400 ptas.)*
- ❑ ref. CD160 *(4.200 ptas.)*
- ❑ ref. ESC24 *(1.300 ptas.)*
- ❑ ref. ESC48 *(1.900 ptas.)*
- ❑ ref. CD-50P *(1.400 ptas.)*
- ❑ ref. CD-15 *(1.300 ptas.)*
- ❑ ref. CD-240P *(1.700 ptas.)*
- ❑ ref. NBC *(3.495 ptas.)*
- ❑ ref. CD-20R *(350 ptas.)*
- ❑ ref. CDLST80 *(2.900 ptas.)*
- ❑ ref. ARCH60 *(2.500 ptas.)*
- ❑ ref. SST-50 *(3.500 ptas.)*
- ❑ ref. SST-30 *(2.900 ptas.)*
- ❑ ref. SST-20 *(2.400 ptas.)*
- ❑ ref. CD-60T *(2.900 ptas.)* Negra
- ❑ ref. CD-120T *(4.700 ptas.)* Negra
- ❑ ref. CDB-80 *(3.900 ptas.)* Negra
- ❑ ref. CDB-160 *(5.900 ptas.)* Negra
- ❑ ref. CD-60TW *(3.200 ptas.)* Marrón
- ❑ ref. CD-120TW *(5.000 ptas.)* Marrón
- ❑ ref. CDB-80TW *(4.200 ptas.)* Marrón
- ❑ ref. CDB-160TW *(6.200 ptas.)* Marrón
- ❑ Gastos envío: 990 ptas.

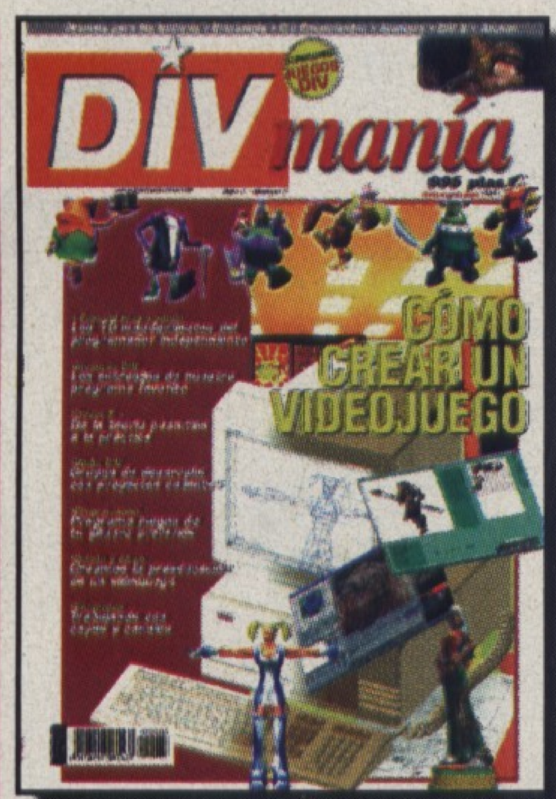
numain
Tlf.: 91 458 20 63
Fax: 91 457 39 04
91 458 68 04
E-mail: numain@tarc3000.com

Nombre...
Apellidos...Teléfono.............................
c/.. Nº..........piso........... letra..............
C.P. .... Localidad..
Provincia...

Forma de pago

Visa: Nº ☐☐☐☐ ☐☐☐☐ ☐☐☐☐ ☐☐☐☐
Trasferencia: 0050 2848 65 0011502322
0049 1359 06 2510019039

DIVmanía



¿Amas la programación de juegos...? Esta es tu revista

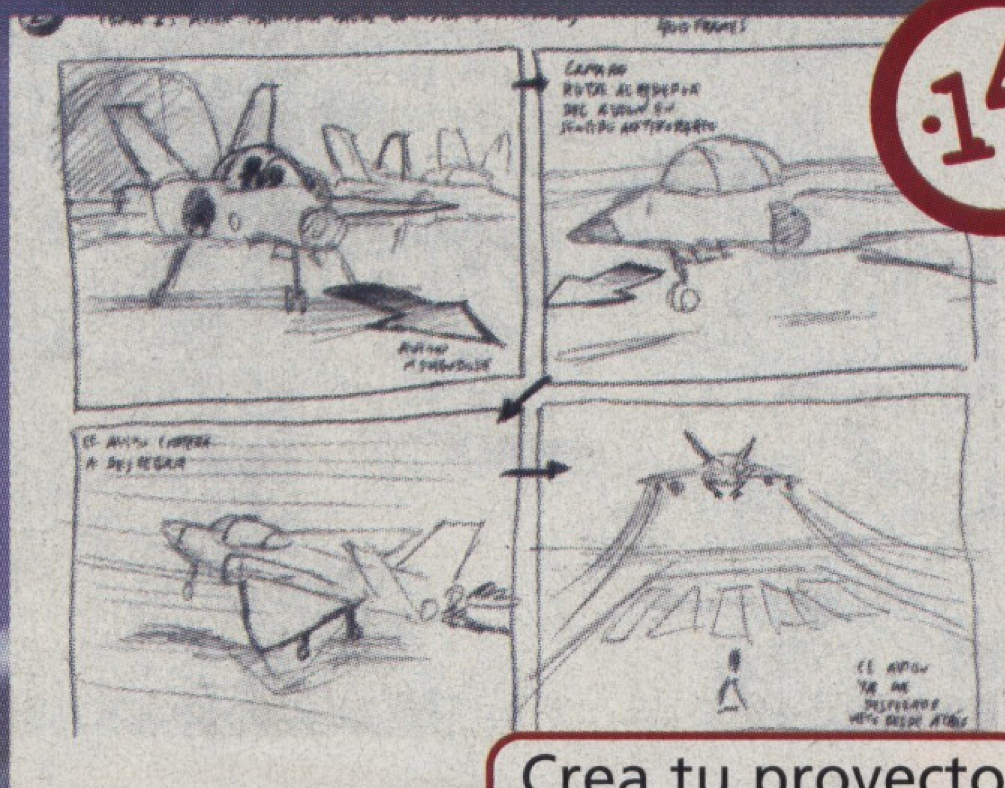
No podemos evitar comenzar estas breves líneas agradeciendo a todos nuestros lectores su apoyo. Las felicitaciones recibidas y las muestras de ánimo a nuestra publicación han sido muchas y eso, como muy bien podéis suponer, es un motivo de enorme satisfacción para todo el equipo que hacemos Divmanía.

Me atrevería a afirmar que hemos iniciado una nueva etapa en la corta andadura de nuestra revista. Un nuevo período en el que, como ya os comentaba en Divmanía 4, una de nuestras prioridades va a ser acudir puntualmente a nuestra cita de cada dos meses, además de seguir ofreciendo lo de siempre: un contenido práctico, ameno y riguroso.

Tampoco podemos obviar una felicitación a todos los que se han animado a participar en nuestro concurso y nos han enviado sus juegos. El número de programas que nos han llegado a la redacción ha sido tal que nos hemos visto en aprietos para decidir cuáles debían ser los ganadores, ¡lástima que no podamos premiarlos a todos! Lo cierto es que la alta participación en el concurso además de la creciente calidad de los juegos, que va aumentando de manera notable, ha hecho que resultara harto difícil decidirse por uno u otro.

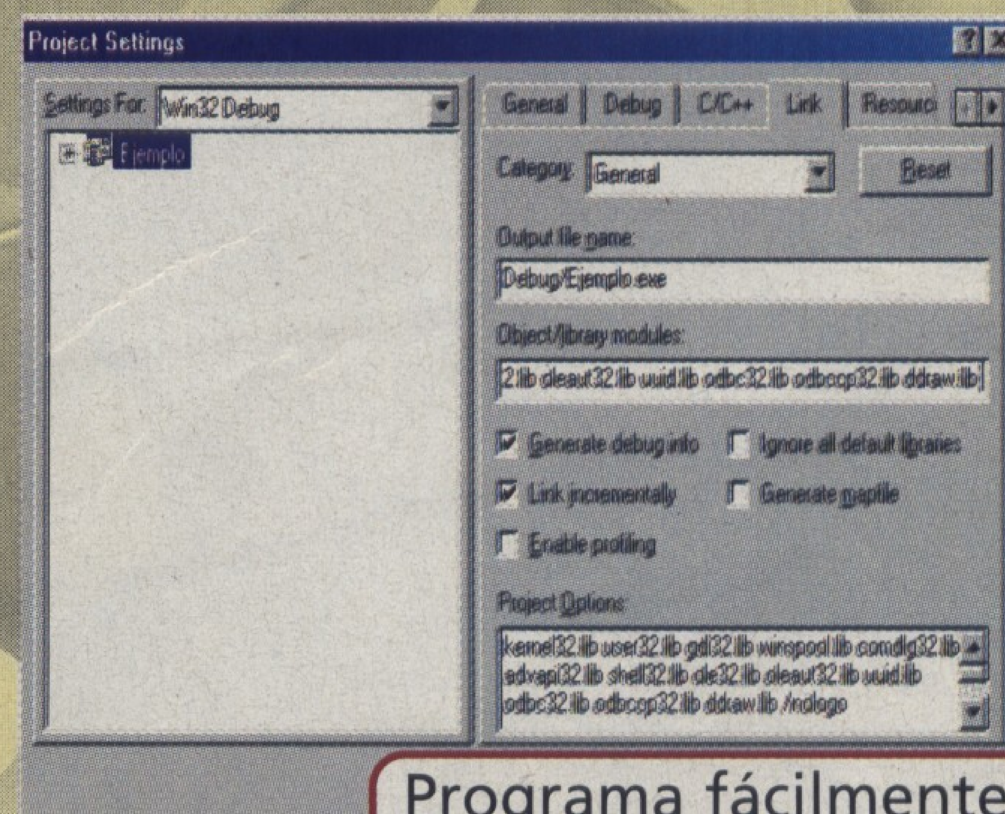
Para acabar quisiera recordar que ésta es una revista abierta y que agradecemos que todos los lectores nos hagan llegar sus juegos, sugerencias, dudas, cartas, críticas, etc. Ya sabéis que para ello cualquier medio es válido y no dudéis en que vuestra participación es muy importante para la revista.

Un saludo. ¡Nos vemos en los kioscos!


Crea tu proyecto


Reportaje especial

DIV Developer



Programa fácilmente

Director: Mario Luis
mluis@prensatecnica.com
Coordinador Técnico: Oscar Condés
gover@prensatecnica.com
Colaboradores: Pablo Trinidad, Sergio Cánovas, Miguel Barroso, David Martínez, Emilio Llamas, Javier Fernández, Fermín Vicente, Armando Vélez, Ramón de España, Carlos González, Raúl Bravo y Jordi Martín.
Edición: Alfredo del Barrio, Guillermo Izquierdo y Martín Moncalvillo

Dirección de Arte: Francisco Calero
Maquetación: Antonio García, José A. Gil, Antonio Barbero, Ana Isabel Madero, Susana Burgaleta y Oscar Menoyo
Portada: Francisco A. Anguís
Publicidad: Marisa Fernández, marisa@prensatecnica.com
Sonia Glez.-Villamil, Noelia Menéndez y Emerio Arena
Supervisión CD-Rom: Alvaro García

Servicio Técnico CD-Rom: Eugenio García
Horario de atención: tardes 16:00 - 18:00 h
E-mail: stecnico@prensatecnica.com
Secretaria de Redacción: Elena Fernández
Departamento de Suscripciones: Sandra Fernández y Noemí Iscart
suscripciones@prensatecnica.com
Departamento de Administración: Juan López, Juan Ignacio Domínguez

Departamento Comercial: Marcelino Ormeño

REDACCIÓN, PUBLICIDAD Y ADMINISTRACIÓN
c/ Alfonso Gómez 42. Nave 1.1.2
Madrid 28037. España
Tfno: 91. 304. 06. 22
Fax: 91. 304. 17. 97
Si llama desde fuera de España, marcar (+34)
E-mail: gover@prensatecnica.com
http://www.prensatecnica.com

Sumario

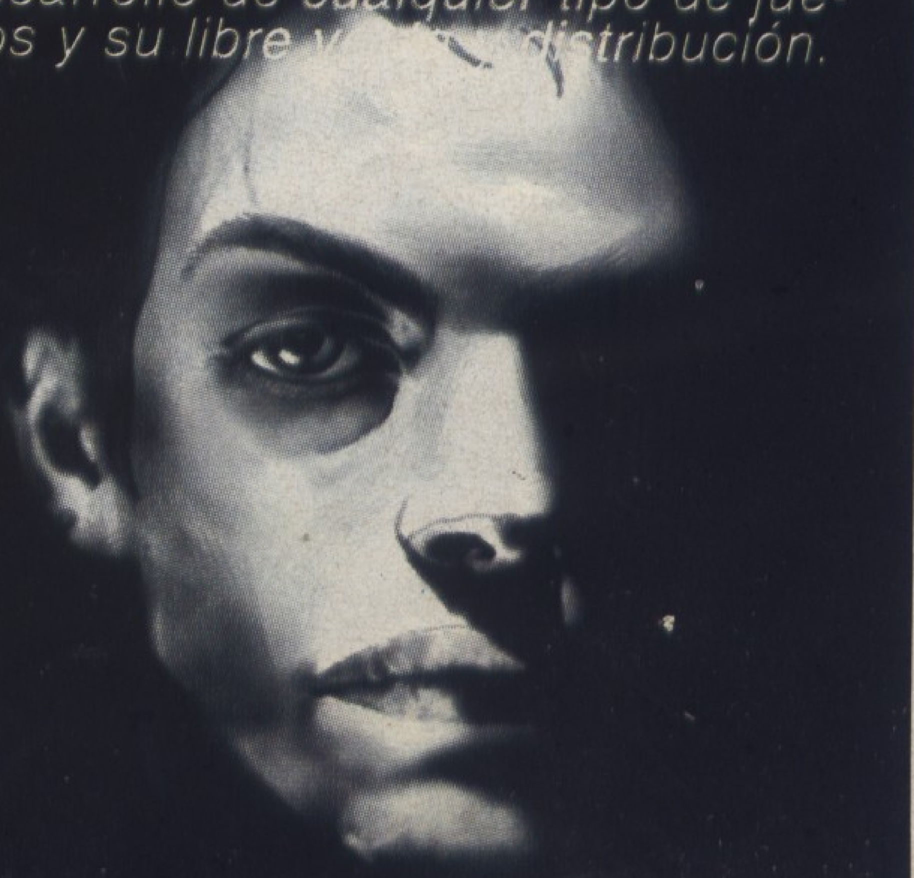
Programas del lector

PREMIAMOS LOS MEJORES JUEGOS DE LOS LECTORES

DIV ha creado toda una escuela dentro del mundo de la programación. Todos los que se han acercado a la herramienta han sido capaces de programar. Por ello realizamos un concurso entre los lectores que hayan realizado juegos con DIV. Os ofrecemos un primer premio de 25.000 pesetas y dos accésit de 20.000 pesetas.

¿QUÉ ES DIV GAMES STUDIO?

DIV Games Studio es una herramienta de programación que facilita en gran manera nuestra inmersión en el software de entretenimiento. Es el primer entorno profesional que permite realizar videojuegos con fines comerciales sin necesidad de un pago adicional. Con el carné de desarrollador incluido se permite el desarrollo de cualquier tipo de juegos y su libre venta y distribución.



Horario de atención al público: de 09:00 a 19:00 h ininterrumpidamente
EDITA: PRENSA TÉCNICA
Director General: Mario Luis
Director Editorial: Eduardo Toribio
Director Técnico: Fernando Escudero
Director de Producción: C.P. Cerezo
Directora Publicidad: Marisa Fernández
Director Comercial: Esteban Martínez
Fotomecánica: Prensa Técnica
Impresión: Pantone, S.A.
Duplicación del CD-Rom: M.P. O., Servicios Ibéricos, Grupo Cóndor
Distribución: SGEL. Avda Valdelaparra, 29
Alcobendas. Madrid

Depósito legal: M-42077-1998
AÑO 1 • NÚMERO 5

El lector lúdico

Sólo para aficionados a los juegos


El coordinador de Divmanía.

En este número nos hacemos eco de una extensa carta. Lo único que sentimos es tener que haberla cortado un poco para conseguir que no ocupara más de una página. Ya veis que no nos ha quedado espacio ni para un comentario de la redacción.

Muy Sres. míos: desgraciadamente, si bien los inventores y artistas tienen la capacidad de crear cosas que generen riqueza, no tienen los medios suficientes para hacer realidad sus proyectos, dependen de los mecenas o empresarios. Por mi parte, como uno más de los muchos cuya labor no es reconocida, he pasado bastante tiempo tratando de convencer a empresarios de que con mis creaciones podrían ganar mucho y pidiéndoles la oportunidad de demostrarlo, sin embargo ello no es posible debido a una gran cantidad de conceptos que se quieren hacer pasar por verdades irrefutables cuando la mayoría de ellos no son más que errores con los que pierden oportunidades estupendas. Quisiera citar aquí **algunas** de esas que yo llamo normas absurdas:

"Se necesita un curriculum" Si entendemos que una persona tiene que demostrar con este requisito qué méritos y éxitos ha tenido para merecer la atención de una empresa, les diré que la política de las empresas, al menos de las que yo he conocido, es que para ser aceptado hay que ser famoso, pero para ser famoso hay que ser aceptado. Creo sinceramente que la mejor evidencia de un autor de juegos son sus obras las cuales ni siquiera merecen la atención de las empresas puesto que ellas ya tienen sus creativos.

"Se necesitan personas jóvenes y emprendedoras" El valor de una persona no se mide con el metro y la balanza, y su talento y capacidad de emprender y realizar proyectos no se mide por su edad, incluso piensa que tener más edad significa tener mucha más experiencia y por consiguiente mucha más efectividad y rendimiento positivo en su trabajo, con esto no se discrimina a las personas jóvenes, antes al contrario, solo se trata de que no se discrimine a quienes las empresas consideran que no lo son. Habría que preguntar a los empresarios que piden personas jóvenes si se han preguntado cual es su propia edad.

"Se necesitan programadores (programador e inventor son la misma cosa)" Falso, inventar y programar son dos profesiones totalmente distintas: 1º) Dentro del mundo del cómic, un personaje como "EL CAPITAN TRUENO" ha sido desde su creación hasta hoy una fuente de beneficios para sus propietarios y, que se sepa, a su creador Víctor Mora nunca se le exigió que lo dibujara ni supiera hacerlo. 2º) Dentro del mundo del cine, las películas de más éxito suelen estar basadas en grandes novelas, y los autores de éstas nada tienen que ver dentro de este medio.



"Un juego ha de estar respaldado por un prototipo informático sólido" Hoy en día una persona sola no puede hacer un videojuego como antes, es una labor que requiere ser realizada con un equipo experto, unos medios materiales adecuados y una fuerza económica abundante. Aunque dicha persona fuera un inventor genial, un informático extraordinario y con medios apropiados, no podría realizar una tarea semejante, y aunque pudiera hacerlo no tendría ni la décima parte de calidad que se exige a un producto destinado a un mercado competitivo. Por ello: A) ¿Qué puede importarle a una empresa que un inventor no presente un juego informatizado si de hacerlo no le serviría para nada y, por otra parte, dicho autor no dispone de una planificación adecuada para hacerlo? B) ¿Resulta lógico que una empresa desestime personas con talento para una especialidad concreta solo porque no reúnen unas características que ni vienen al caso ni son necesarias?

"Las empresas disponen de sus propios inventores" Si esto fuera una verdad indudable, habría que preguntar a las empresas, sobre todo a las más importantes y poderosas, porqué gastan cientos de millones pagando a agencias de publicidad a cambio de que éstas les proporcionen ideas con las que puedan lanzar sus productos al mercado.

Finalmente, como he dicho y repito, sería necesario escribir un libro para enumerar todas las llamadas normas absurdas, pero espero que, si estas líneas "salieran a la luz", haya lectores que estén de acuerdo con ellas y lo expresen con sus cartas a la redacción de la revista.



Prensa Técnica presente en el SIMO

Como podéis ver por la foto que acompaña a esta noticia la empresa editora de esta revista, Prensa Técnica, no falto a la cita anual en la, hoy por hoy, primera feria a nivel nacional dedicada a la informática y las telecomunicaciones.

En el SIMO estaban presentes las empresas más punteras del dinámico sector de la información y las comunicaciones, hardware, software, publicaciones técnicas, empresas de telefonía, distribución, Internet, etc. Lo último en tecnología informática.

La presencia más fuerte estuvo a cargo de los mayoristas de componentes de ordenador, impresoras, servidores, tarjetas gráficas, consumibles, accesorios, etc. En cuanto a software, Microsoft presentó su *Windows 2000*, como otros fabricantes que han elegido el próximo año para que dé nombre a la nueva versión de sus productos.

En cuanto a juegos destacó el pabellón de Havas que presentaba el lanzamiento de sus últimos trabajos: *Faraón, Homeworld, Gabriel Knight III, Diablo 2* o *Edgar Torronteras Extreme Biker.*



También tuvieron una fuerte presencia los stand dedicados a promocionar portales de Internet como Navegalia, Ya.com y más. En resumen, un escaparate imprescindible para todos los profesionales y aficionados a la informática.

Ancient Evil, nuevo juego de Hammer

Digital Dreams, filial de Hammer Technologies, los creadores de *DIV* y *DIV 2*, van a lanzar próximamente al mercado el juego *Ancient Evil*. A este juego se le puede clasificar dentro del género del rol, y tendrá una mecánica parecida a *Diablo*, o sea rol pero con un alto componente arcade. Son necesarios reflejos rápidos para salir indemne de esta aventura ambientada en la tenebrosa Edad Media.



Magia, combates cuerpo a cuerpo, mazmorras, castillos y laberintos es lo que nos espera en *Ancient Evil*. El objetivo del juego será intentar penetrar en un lugar llamado "Cripta de los Antiguos", un mundo subterráneo construido hace más de 5000 años y que sólo dos personas han conseguido recorrer y vivir para contarlo: Alaric, que se ha convertido en el "casero" del lugar, y Jetraal.

Alaric ofrece una recompensa a todo aquel que se adentre por los pasadizos subterráneos y logre sobrevivir a las miríadas de seres que pululan por allí. El espíritu de Jetraal, muerto unos siglos antes, avisa a los incautos de los peligros que se hayan ocultos en las mazmorras. El juego presenta unos gráficos que lo hacen bastante interesante si sale a buen precio, cosa que así parece que va a suceder.



Dreamcast se hace un hueco en el mercado

Si el mercado de los juegos para ordenador no para de crecer, la penetración de las diferentes plataformas consoleras no le va a la zaga. La nueva consola de Sega, la Dreamcast, está teniendo bastante éxito en España. Se esperan vender unos 100.000 aparatos de aquí a final de año. Hoy por hoy es la más seria rival de la PlayStation de Sony, que hasta hace poco copaba el mercado de este tipo de productos.





Y claro, estas distintas plataformas demandan juegos y más juegos para alimentar sus voraces entrañas de silicio. Y eso sin contar que próximamente saldrán a la calle la PlayStation 2, la consola Dolphin de Nintendo, y probablemente Microsoft esté desarrollando su propio aparato para competir con estos gigantes japoneses. Toda una oportunidad para los programadores de videojuegos.





Juegos "estrella" para las Navidades

Como suele hacer la industria cinematográfica, las Navidades son una de las fechas clave para que las compañías distribuidoras de videojuegos pongan de largo sus productos más conocidos. Estas fiestas navideñas del fin del milenio no son una excepción, y es que esta época del año es cuando más dinero se gasta en este tipo de productos lúdicos, y esto las empresas lo saben de sobra.





Así, se espera un aluvión de títulos de software de entretenimiento, muchos de ellos segundas, terceras o cuartas partes de juegos de sobra conocidos. Es curioso ver como, a diferencia de las películas de cine, en la industria de los videojuegos se puede decir casi siempre que "segundas partes siempre son buenas", esto es debido a que los avances tecnológicos en hardware enseguida se traducen en el software.

Juegos que saldrán en estas fechas son la cuarta parte de *Tomb Raider*, con la incombustible Lara Croft esta vez por tierras del antiguo Egipto, intentando descubrir el misterio que envuelve estas milenarias piedras. Parece que esta entrega tendrá bastantes más ingredientes de aventura gráfica que las anteriores, veremos si Lara sabe darle tan bien al "coco" como apretar el gatillo.

La segunda parte de *Age of Empires*, que se subtitulará *Age of Kings*, es otro de los juegos más que recomendables para invertir nuestro siempre escaso peculio. Los gráficos de este clásico de la estrategia en tiempo real han sido increíblemente mejorados, los edificios construidos a escala casi real de los soldados crearán escuela, y si no al tiempo.





Otros juegos que se lanzarán en diciembre y que venderán grandes cantidades de copias serán títulos tan conocidos como *Diablo II, Fifa 2000, Unreal Tournament* y *Quake 3*. Todos ellos prometen mejorar la estupenda calidad de sus anteriores entregas. Todo un paraíso para los "jugadores de ventaja".









Ubi Soft asalta al sudeste asiático

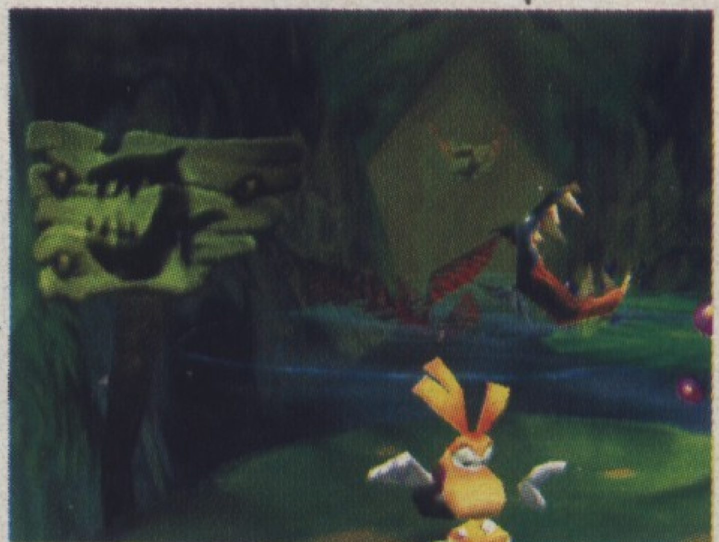
La multinacional de desarrollo y distribución de videojuegos Ubi Soft acaba de abrir una nueva filial en Hong Kong para meter la cabeza en el jugoso mercado asiático, uno de los que experimentan un más alto crecimiento del mundo.

Esta nueva sede intentará promover la difusión de sus productos en todo el continente, especialmente en las regiones de Taiwan, Singapur, China, Tailandia y Malasia. No solamente esperan distribuir juegos de desarrolladores occidentales, también esperan conseguir acuerdos con empresas creadoras y distribuidoras de videojuegos de esa parte del mundo para lanzarlos en Europa y EEUU.



La experiencia previa que ha tenido la empresa con el juego de estrategia *Seven Kingdoms II*, que ha sido creado por Enlight Software, una empresa con sede en Hong Kong, les ha animado a emprender esta aventura comercial. También el éxito de alguno de sus juegos en estos mercados, *Raiman 2, the Great Escape* ha conseguido vender 700.000 copias en Asia, les ha convencido de que Ubi Soft debe incrementar más su presencia en el Sudoeste Asiático.





Esperemos que podamos gozar próximamente de los productos desarrollados por los creativos de estas lejanas tierras, ya que raramente se distribuyen en Europa si no son títulos de éxito.

Eidos

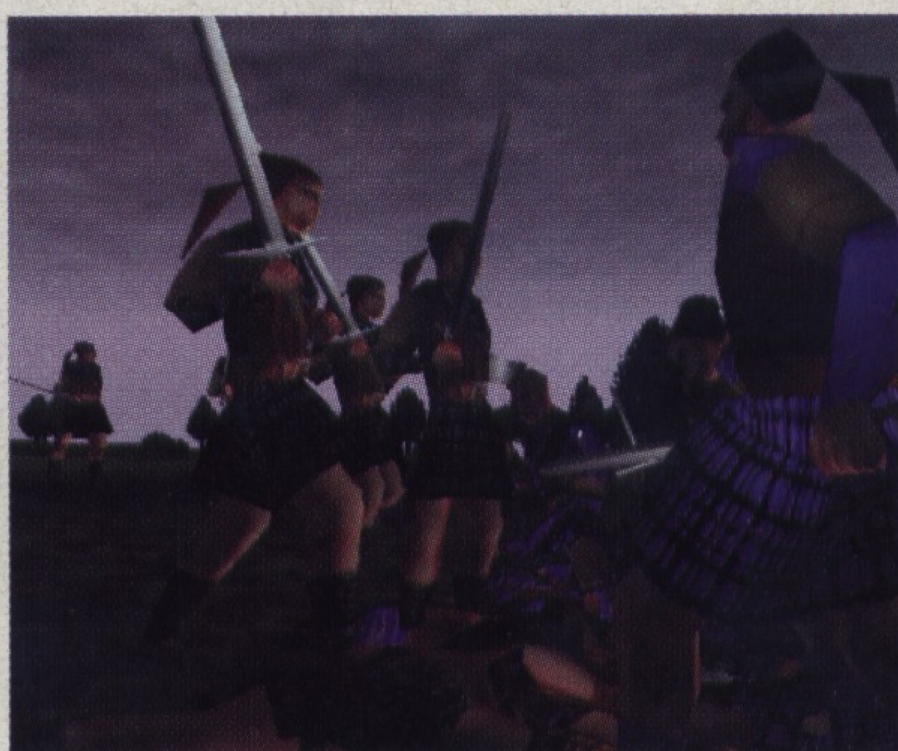
Compañías con acento inglés

Nuestro ciclo de reportajes dedicados a las compañías desarrolladoras y distribuidoras españolas de software lúdico ha llegado a su fin. Ahora toca el turno de hablar de las empresas extranjeras y hemos elegido a Eidos para que rompa el hielo en esta nueva etapa.

Para hablar de Eidos hemos de remitirnos a Eidos PLC, la entidad que engloba a Eidos Interactive, Eidos Technologies y Glassworks, un formidable trío que abarca desde la producción de videojuegos a la investigación de nuevas tecnologías aplicadas a los mismos.

Todo comenzó hace casi una década, bastante tiempo si tenemos en cuenta los cambios que se producen en el mundo de la informática de año en año. Por aquel entonces, Eidos, se componía de un reducido grupo de trabajadores que llevaron a cabo un proyecto que ha terminado de consolidarse hace apenas veinticuatro meses, durante los que se han dado los principales cambios en la compañía. Ahora la plantilla ha crecido hasta la escalofriante cifra de 500 empleados y sus programas, sobre todo en el ámbito de los videojuegos, se comentan en la prensa especializada meses antes de su salida al mercado.

Eidos Interactive es una desarrolladora de entretenimiento para los soportes más extendidos: PlayStation, Nintendo 64 y PC. Tiene oficinas en Londres, París, Hamburgo, Tokio, San Francisco y



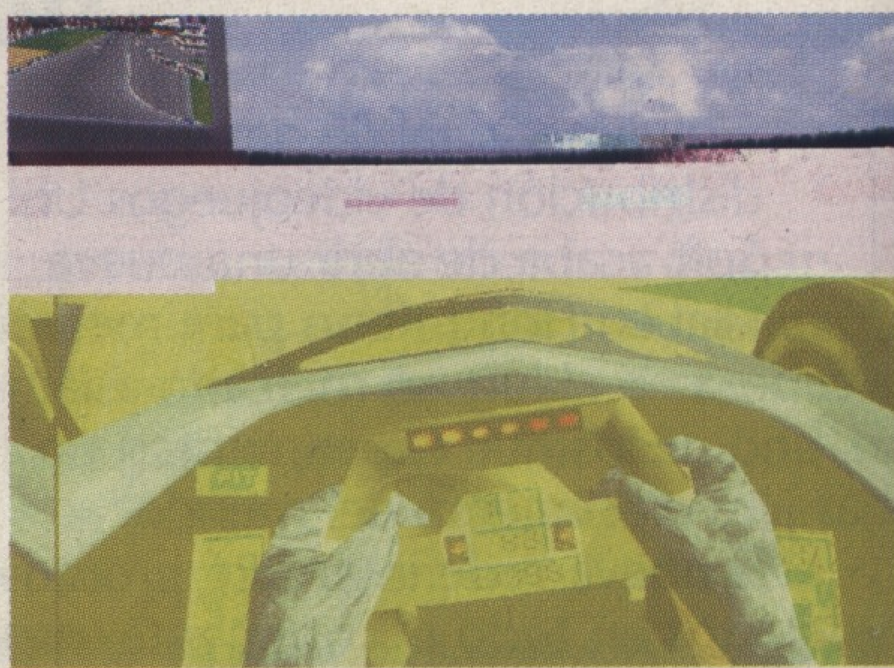
Singapur, lo que da una idea muy precisa de la amplitud de cada uno de sus lanzamientos. Presumen, y no sin motivos, de centrarse en la jugabilidad y en los elementos innovadores de cada uno de sus productos, aunque no han pasado por alto la oportunidad de explotar sus series más famosas: *Championship Manager* y *Tomb Raider*.

Lara es la mujer cibernética más famosa del mundo que asoma sus exageradas curvas en cuanto alguien escribe o dice "Eidos". Y es que se ha convertido en el emblema de la compañía y en la mejor embajadora de la mayor aportación de ésta al mundo de los videojuegos, la perspectiva en tercera persona, que ha calado tan hondo en la industria del entretenimiento como en su día lo hiciera la subjetiva, con el famoso *Doom* de Id. Por supuesto, no sólo de tecnología vive este juego y es sabido por todos que el carisma de su protagonista ha jugado una importante baza al respecto, como lo demuestra el acuerdo con Paramount Pictures para sacar una película basada en las aventuras de esta heroína que encabeza una saga calificada de "Producto del Milenio" en el mundo del entretenimiento. Asimismo, Eidos Interactive ha firmado acuerdos con Michael Owen e ISM (International Sports Multimedia, para publicar el juego *UEFA League Championship*) y ha adquirido los derechos de la película "Braveheart" de Paramount Pictures y Twentieth Century Fox.

En cuanto al desarrollo de los diferentes productos, Eidos Interactive apuesta por el trabajo de pequeños, pero bien motivados, grupos autónomos. De hecho, tiene acuerdos de diferentes grados con 21 desarrolladoras. Algunas de éstas pertenecen completamente a Eidos, pero la mayoría de las relaciones con las mismas se basan en acuerdos a largo plazo, lo que generalmente incluye un mínimo de tres o cuatro lanzamientos.

En cuanto a las filiales tenemos a Eidos Technologies que es otra de las subsidiarias de Eidos y permanece como una derivada de la compañía fundada en 1990, aunque no se dedica al desarrollo de videojuegos, sino a la investigación, concretamente al estudio de vídeo telefonía, CD-Rom vídeo, códigos de comprensión para vídeo y todo lo relacionado con la tecnología de redes e Internet.

Glassworks fue adquirida por Eidos en Marzo de 1996. Es una compañía de post producción











especializada en dar el repaso final a los efectos visuales digitalizados usando máquinas de Silicon Graphic y otras técnicas digitales.

Y ahora vamos a pasar a hablar de los juegos más emblemáticos que ha sacado la compañía en los últimos tiempos. Empezamos por *Braveheart*. Es de agradecer que el juego no descanse sólo en la fama de la película, ya que sus creadores han desarrollado un programa con interesantes innovaciones en dos de los campos más populares de los últimos tiempos: la estrategia y los entornos 3D. Con respecto a la primera, destaca el hecho de que combine estrategia en tiempo real, para los combates, con una mecánica de turnos para las fases de planificación y gestión de recursos. El jugador tiene la opción de escoger uno de entre los dieciséis clanes escoceses disponibles. A partir de ahí podemos tomar el camino de la diplomacia o la guerra para unirnos contra los ingleses o enfrentarnos a éstos con nuestras propias fuerzas. En este juego se han incluido personajes de la película (fielmente reproducidos en 3D) y videoclips de la misma. Otra de sus virtudes es la de contar con un mapa, también reproducido con exactitud en 3D, de Escocia e Inglaterra. Una combinación de elementos de otros títulos pulida con la mejor tecnología y que ofrece al jugador un importante margen de decisiones.

Otro juego que será noticia próximamente será *Daikatana*. John Romero, creador de las series *Doom* y el primer *Quake*, está detrás de este futuro lanzamiento. No puede haber referencia más positiva tratándose de un *shoot 'em up* en primera persona que trasladará al jugador a ambientes tan dispares como el Japón del año 2455, la Grecia Clásica del año 1200 A.C, Noruega en el año 560 A.C y un San Francisco futurista de principios del siglo XXI. Por supuesto, esta variedad afectará a las armas, el arte, la música y las criaturas de cada época.

A pesar de ser uno de esos juegos en los que primarán los reflejos y la puntería sobre cualquier otra cosa, se van a incluir detalles interesantes que lo distinguirán de la competencia, como el que Hiro, el protagonista, pueda entablar diálogos con sus compañeros controlados por el ordenador, Superfly Johnson y Mikiko Ebihara. Y, como es de esperar, el aspecto multijugador se ha cuidado en extremo, como demandan los usuarios de este género. Posiblemente estamos ante el único rival de *Quake III*.

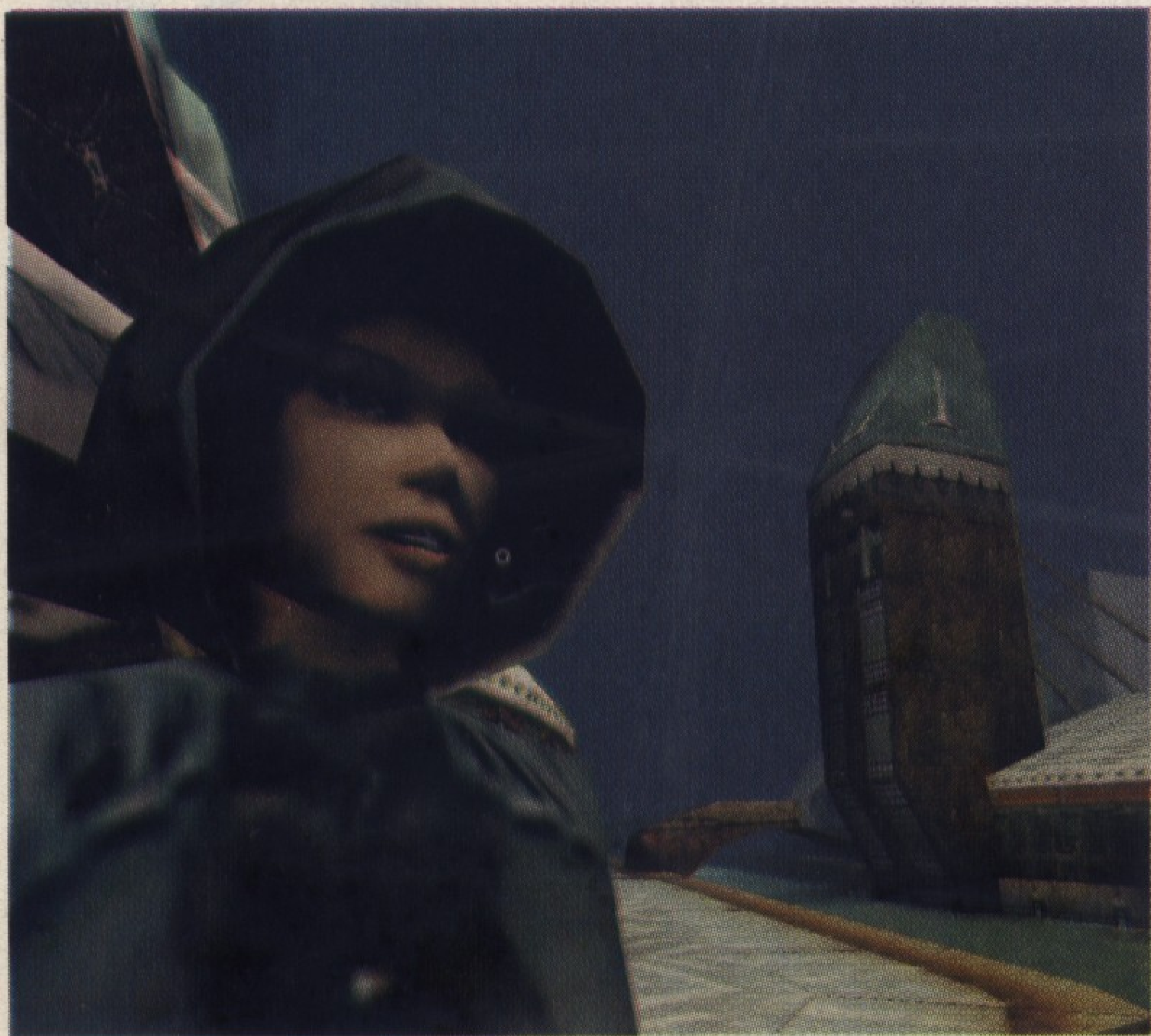
Tomb Raider ha sido la principal causa de que Eidos haya podido llevar a cabo las extraordinarias inversiones en tecnología y nuevos proyectos

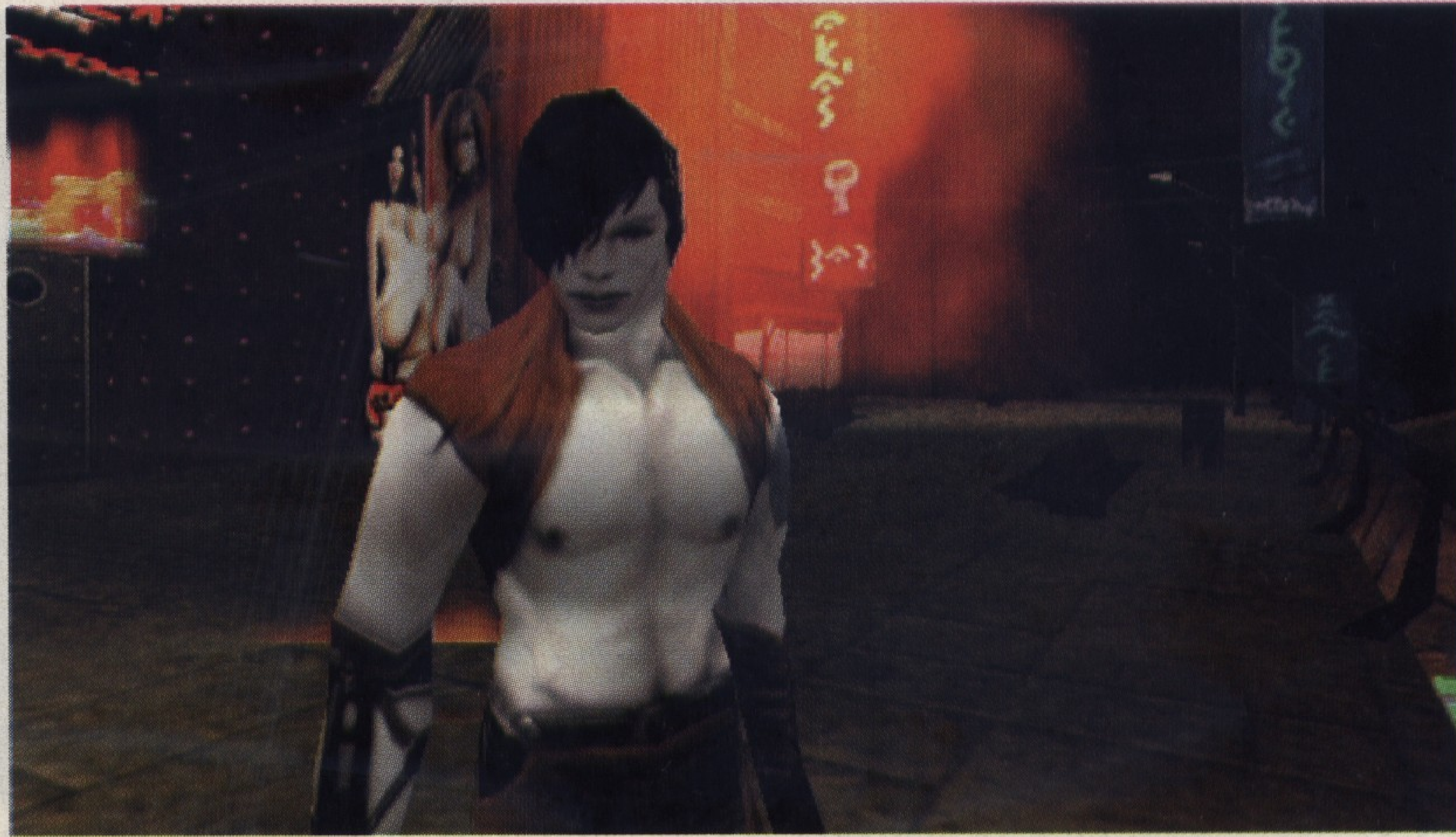
También tenemos que hablar de *Nomad Soul*, una mecánica revo-













lucionaria en cuanto al desarrollo de cada partida y una estética futurista propia del mejor cómic de calidad, avalan el lanzamiento de un juego al que se ha apuntado incluso el Duque del Rock, David Bowie, que se ha accedido a ser "modelado" en 3D para aparecer en la impresionante ciudad en la que se desarrollará el juego. El jugador, aquí radica la principal innovación, podrá encarnarse en los diferentes habitantes de una extraña dimensión a la que será arrojado sin un objetivo en concreto, por lo menos al principio.

La mayoría de los nuevos lanzamientos de esta compañía se han adherido triunfalmente a la moda de las 3D

Explorar cada calle, el interior de los edificios, montar en vehículos futuristas, entablar diálogos con otros habitantes e investigar la razón que lo ha llevado hasta allí, serán los elementos que amenicen cada partida en un juego que admite una completa libertad de acción a lo ancho y largo de una ciudad levantada en 3D. Ciclos de día y noche y combates y movimientos faciales en tiempo real, serán otros elementos que añadirán vistosidad al producto por medio de un sentido de innovación a dos bandas: tecnológica y argumental. Un "juego total" en el que pasaremos del arcade al simulador, pasando por las plataformas o las luchas callejeras.

A los que le gustan los juegos de carreras de coches es posible que *Formula 1* sea su título preferido. Basado en la temporada de 1998, se presenta este simulador de última generación, en el que se ha puesto especial interés tanto en la física y comportamiento del coche, como en la base de datos cuyo fin es llevar al jugador a un entorno lo más realista posible, como lo demuestra el que cada uno de los 22 coches participantes y los 16 circuitos disponibles están modelados de forma individual.

Entre sus características más sobresalientes cabe destacar el esfuerzo que se ha hecho para que la versión del juego por software esté a la altura del versiones aceleradas con una tarjeta 3D, los veinte ángulos de cámara, el soporte para 12 jugadores en Red, los 3 niveles de dificultad, las diferentes condiciones climatológicas, etc.

Y nos queda hablar de la que será la cuarta parte del mascarón de proa de la compañía: *Tomb Raider.* Este entrega tendrá un carácter más marcado de aventura gráfica y se desarrollará íntegramente en Egipto. Allí Lara tendrá que resolver una oscura trama que tiene que ver con los conocimientos de los antiguos sacerdotes de esta tierra de faraones y construcciones fabulosas. Estamos seguros que has jugado o que jugarás con alguna obra de esta compañía.





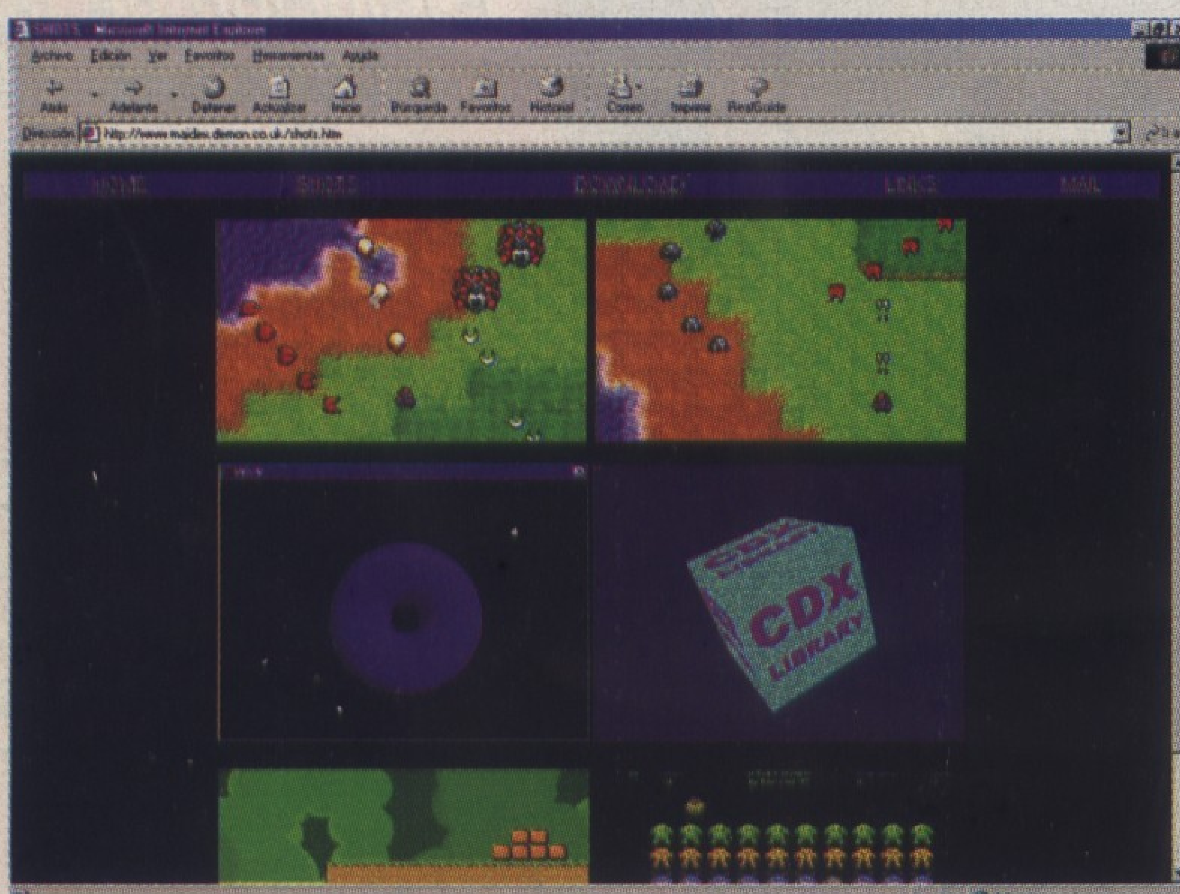


Cómo crear un videojuego

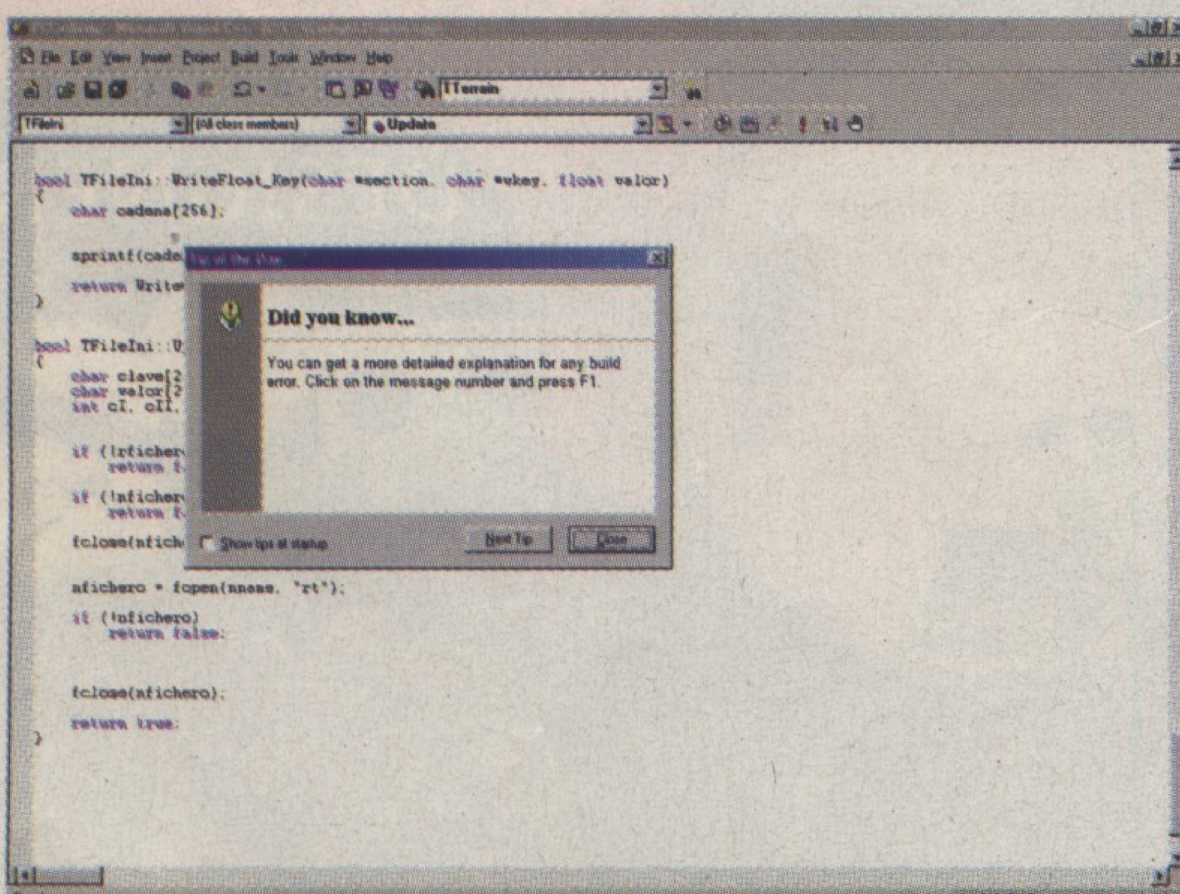
Lo que debes hacer para programar tus juegos

Si estás cansado de jugar con las obras de otros y se te ha pasado por la cabeza alguna buena idea que podría ser el origen de un gran videojuego, en este artículo te damos las claves que debes seguir para llevar a la práctica tus aspiraciones de creación lúdica.

Mediante este artículo, pretendemos dar una pequeña orientación de los pasos que deberíamos seguir para desarrollar nuestro propio videojuego. No pretendemos sentar ningún dogma, ni que se sigan nuestras recomendaciones al pie de la letra, si no, más bien, dar unas normas básicas que pueden ayudarnos a que nuestro juego llegue a buen puerto.



CDX es una librería para usar las DirectX sin complicarnos la vida. http://www.maidex.demon.co.uk.



Visual C++ es la herramienta preferida de los programadores para usar las DirectX.

Primeros pasos

En primer lugar, hay que tener clara la idea argumental del juego. No necesitamos ser unos escritores consumados, tan sólo tener la suficiente imaginación para plantear el hilo o guión del juego. Decidir el protagonista, el género, las diferentes misiones, etc. Si nos es posible podemos hacer un "story-board" del juego, así como las posibles intros que deseemos tener en el mismo.

El sistema de visualización también debemos decidirlo en un principio. Básicamente es cómo vamos a mostrar nuestro juego, en perspectiva isométrica, cenital, 2D, 3D, etc. Cada una de estas formas de visualización tiene requerimientos diferentes, y diferentes retos técnicos.

El equipo de trabajo que va a realizar el juego debe ser previsto desde el inicio. Cuántas personas necesitamos para construir el juego, cuántos programadores y cuántos grafistas. Esto es muy difícil de evaluar en este punto del desarrollo, pero cuanto más nos ajustemos ahora, más fácil será afinar nuestro producto. Hemos de tener en cuenta que cuántas menos personas intervengan en el desarrollo, menos gastos y más ingresos por persona, pero esto supondrá una mayor carga de trabajo sobre cada componente del grupo de desarrollo, así como una demora considerable en el plazo de entrega.

Debemos ajustar el equilibrio: tiempo de desarrollo-equipo de trabajo. Nuestro consejo es que sigamos al pie de la letra el viejo refrán "La avaricia rompe el saco". Algo que tenemos que tener muy presente, es que no es más importante el programador o el grafista, en ningún caso, si queremos obtener un producto de calidad no debemos infravalorar ninguno de los aspectos técnicos. En caso contrario, podemos tener un producto que desde el punto de vista de la programación, e incluso desde el punto de vista argumental, sea realmente extraordinario, pero que pase sin pena ni gloria al no tener unos gráficos vistosos. Y a la inversa, podemos crear un juego con unos gráficos extraordinarios pero terriblemente aburrido, o flojo desde el punto de vista de la programación.

La palabra clave es equilibrio, y consenso. A pesar de que debemos tener al menos un coordinador, un jefe, es recomendable que establezcamos un consenso en el equipo, que todos participen en el desarrollo del juego, en el hilo argumental, sugiriendo nuevas ideas que aporten nuevas posibilidades. Debemos pensar que el desarrollo de un juego normalmente es un proceso que lleva varios meses, cuando no años, y que alguien del equipo que no se integre, que no vea que el juego es algo suyo en cierto modo, puede provocar retrasos y discusiones que debemos atajar cuanto antes, y la mejor forma de evitar estos posibles conflictos es lograr que todos consideren el proyecto como algo propio.

Planificar las tareas

Uno de los pasos que debemos seguir rigurosamente si no queremos ver nuestro proyecto plagado de retrasos y de trabajo incompleto, es planificar cuidadosamente las tareas que vamos a realizar, el tiempo que vamos a emplear en cada tarea, y el orden en que vamos a realizarlas. Nunca podremos saber con exactitud el tiempo que emplearemos en realizar cada tarea, quizás en una rutina tardemos más de lo previsto, y en otra menos. Un gráfico puede ser más complejo que otro, pero en cualquier caso

debemos intentar aproximarnos lo más posible, siempre más por exceso que por defecto. Es preferible decirle a nuestro posible comprador o cliente que nuestro proyecto estará finalizado en cuatro meses y que nos sobre un mes, que vender la moto, decir que estará finalizado en tres meses y tenerlo en cuatro.

Para realizar nuestro diagrama de tiempo podemos emplear distintas técnicas, si queremos ser un poco profesionales y conocemos las técnicas, podemos emplear diagramas Pert o Grant, pero si no tenemos estos conocimientos bastará con que nosotros mismos nos hagamos un calendario completo de las tareas a realizar y el tiempo que emplearemos en hacerlas. También podemos utilizar programas preparados para esta planificación, desde luego, siempre obtendremos un esquema bastante más claro y que podremos modificar sobre la marcha, pero no es imprescindible invertir nuestro escaso presupuesto en este tipo de software.

Sistema operativo

Otro de los puntos fuertes que condicionarán nuestro proyecto es decidir el espacio de trabajo del juego, es decir, el sistema operativo que lo sustentará. *MS-DOS, Windows, Linux,* cada uno de ellos tiene distintos requerimientos y distintas herramientas, por lo que es una decisión prioritaria establecer rápidamente el sistema operativo. Lo normal es que nos decantemos por *Windows*, esta plataforma es la más extendida y cuenta con numerosas herramientas tanto para el programador como para el diseñador gráfico.

Windows y *DirectX* serán, sin duda, nuestra elección en la mayor parte de los casos, su implantación es mayoritaria y podremos asegurarnos así que nuestro producto va a tener la máxima difusión.

Librerías

El número de librerías para creación de juegos 2D y 3D es enorme, unas de mayor calidad que otras, la elección siempre debe ser nuestra. Facilidad de uso, precio, disponibilidad de soporte, servicio técnico, deben ser las razones que nos hagan decidirnos por alguna de estas librerías, o por ninguna.

Siempre podemos empezar desde cero, es una opción como otra cualquiera, pero nos llevará más trabajo, aunque seguramente tendremos lo que queramos. O tal vez no, depende de nuestros conocimientos, puede que algo que creímos en un principio asequible llegue a ser una autentica pesadilla, y los retrasos, auténticos enemigos de la creación de un videojuego, sean habituales y contínuos.

Elegir las herramientas

Debemos ante todo tener en cuenta que el planteamiento que hagamos de nuestro juego nos hará que nos decantemos por una u otra herramienta. Por tanto, así como hemos planificado cuidadosamente nuestro guión, éste es el momento de definir cómo vamos a representar nuestro juego. Un juego 3D tiene unos requerimientos completamente distintos de uno 2D y, por tanto, emplearemos diferentes herramientas.

Si nuestro juego es claramente 2D, en perspectiva cenital o isométrica quizás debamos decantarnos por herramientas netamente 2D. No es que no podamos emplear herramientas destinadas a un uso 3D, sino que mediante una herramienta especifica terminaremos nuestro proyecto antes y con una mayor eficacia.

Para realizar un buen juego es necesario que planifiquemos las tareas cuidadosamente

Lo habitual hoy en día, es emplear herramientas de carácter general, que no se limitan a un campo concreto, pero si necesitamos aprender a manejarlas, quizás decantarnos por una herramienta especifica sea un acierto a corto plazo.

Para *Windows* y *DirectX* la mejor herramienta para el programador es *Microsoft Visual C++*, con la versión 7.0 de *DirectX*. Tenemos acceso a estas desde *Visual Basic*, pero la opción indiscutible es *Visual C++*. Es más, la mayoría de las librerías que podemos utilizar de interfaz con *DirectX* han sido hechas por y para *Visual C++*.

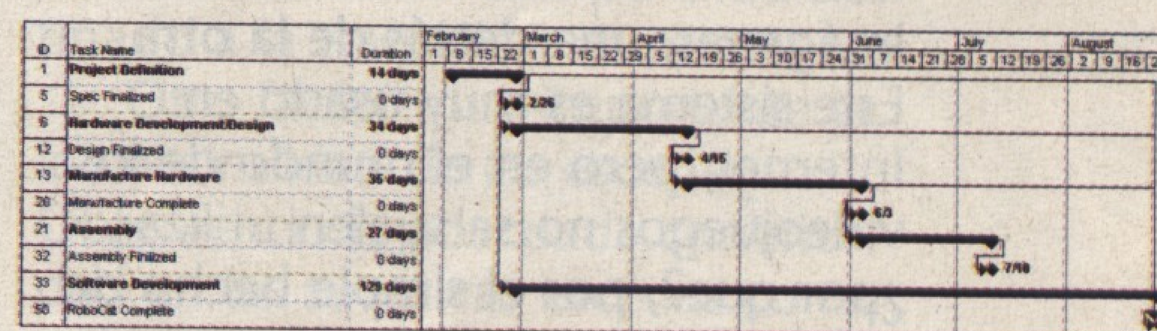
Otra cosa, es que decidamos desarrollar nuestro juego para DOS. En ese caso, quizás utilizar otro sistema como puede ser *DIV*, pueda llegar a ser una opción bastante aceptable, todo depende de las posibilidades que queramos darle al juego, y del nivel técnico que poseamos.

Como organizar el material gráfico

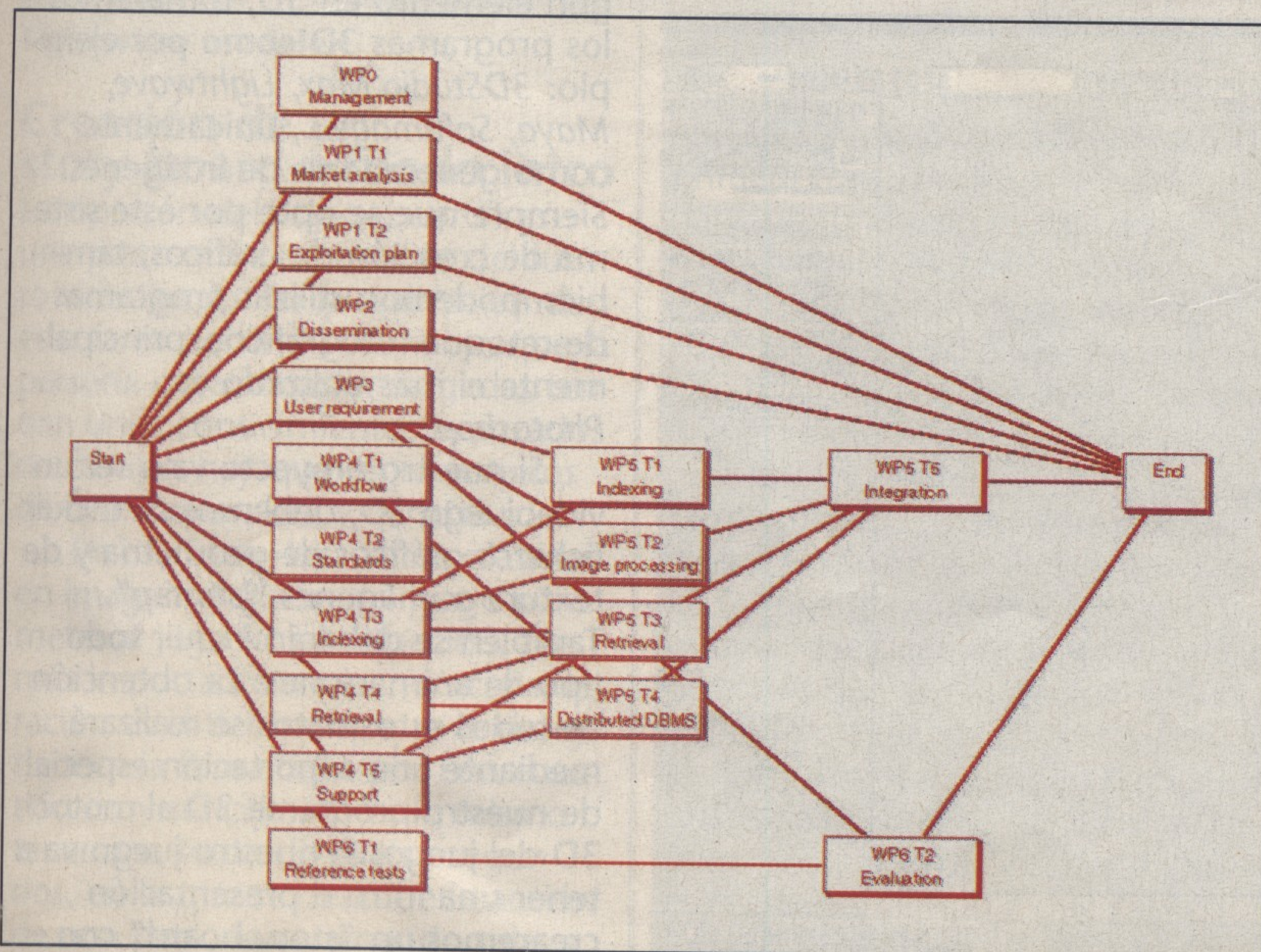
En todos los proyectos el orden es un elemento fundamental, en nuestro desarrollo el apartado gráfico debe de cumplir unas funciones



DIV2 nos permite hacer sin grandes complicaciones un juego de nivel medio. http://www.divgames.com/es/index.htm .



El diagrama GANT es otra forma de definir nuestro trabajo, esta técnica nos asegura una perfecta planificación temporal.



Un diagrama de PERT nos permite planificar el orden y el tiempo de cada tarea.

muy básicas y de perfecto entendimiento con las otras tareas. El departamento o grupo encargado del material gráfico colabora en un proyecto común, esto hay que tenerlo muy en cuenta.

Todos los gráficos se integran bajo programación, los gráficos pueden ser muy diversos y con gran variedad de formatos. Por ejemplo tenemos formatos de videos "*.avi, *.mpg, *.mov, *.fli, *.flc...", también tenemos imágenes "bitmaps". Este tipo de ficheros, por lo general, suelen ser imágenes estáticas, por ejemplo los ficheros *.gif animados permiten realizar una animación mediante la sucesión de imágenes, una detrás de la otra. Este sistema es muy usado en Internet, pero en el mundo de los videojuegos no se suelen utilizar, ¿por que?, por el simple hecho de la integración de los gráficos mediante programación, de esta forma se pueden repetir imágenes sin tener que estar utilizado este tipo de ficheros de poca calidad gráfica.

Los ficheros más utilizados hoy en día en el mundo de los videojuegos son los ficheros *.bmp, *.pcx, y *.tga. El más sencillo de carga en los juegos son los ficheros *.pcx. La peculiar característica de los ficheros *.tga es la incorporación de un canal adicional, llamado canal "Alpha". Este canal se utiliza especialmente para las trasparencias. Muchos de vosotros os preguntaréis cómo se realizan las transparencias en todos aquellos formatos que no permiten canal "Alpha" de transparencia. La forma es bastante sencilla, se utiliza un fondo con un color liso y uniforme, este color mediante programación se definirá como transparencia, de esta forma solucionamos la carencia de canal de transparencia adicional. También existen muchísimos ficheros que son propios de los programas, todos estos ficheros son los que podríamos llamar ficheros generadores o fuentes.

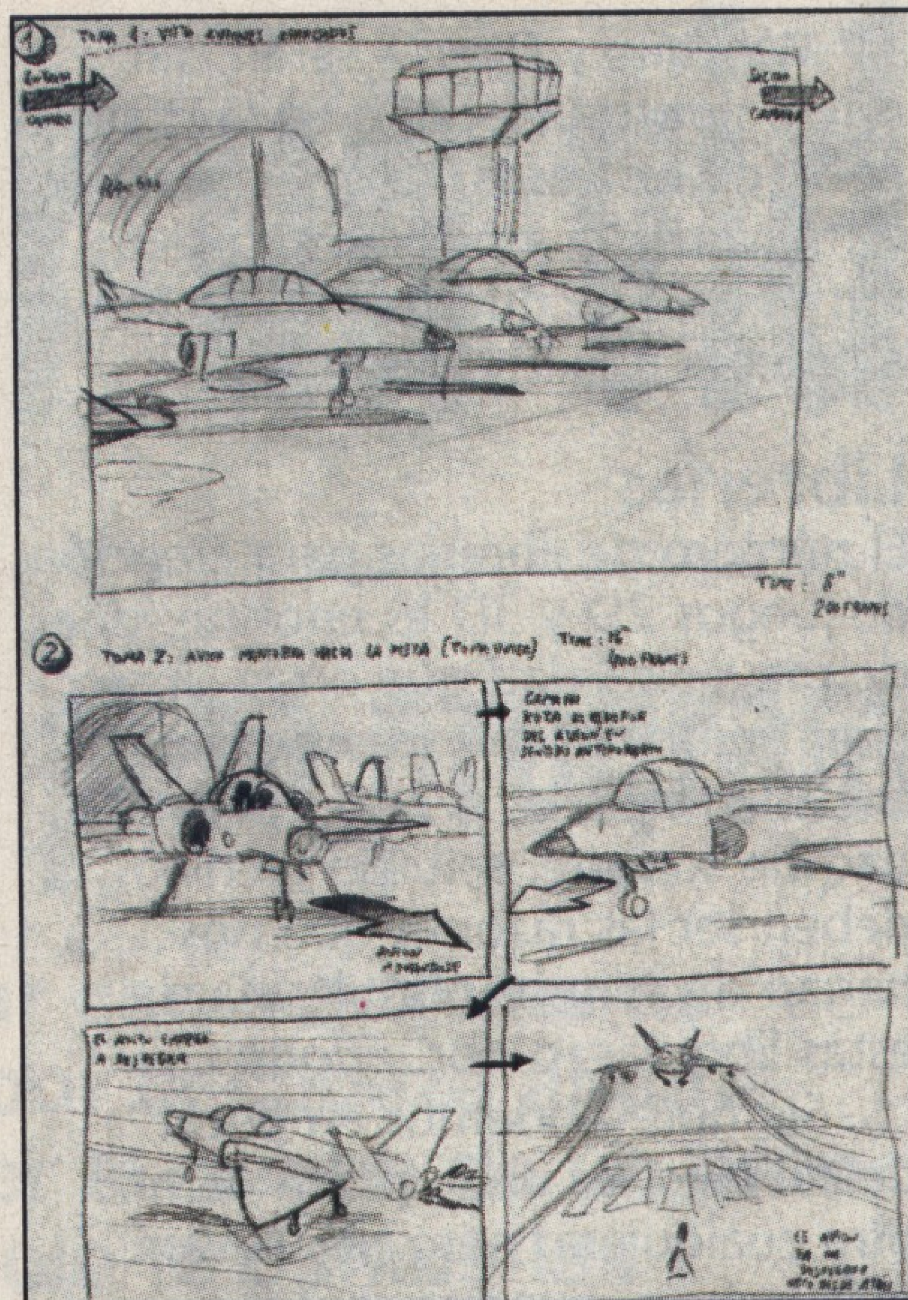
Es importante desde el principio saber qué es lo que queremos hacer y cómo vamos a realizarlo

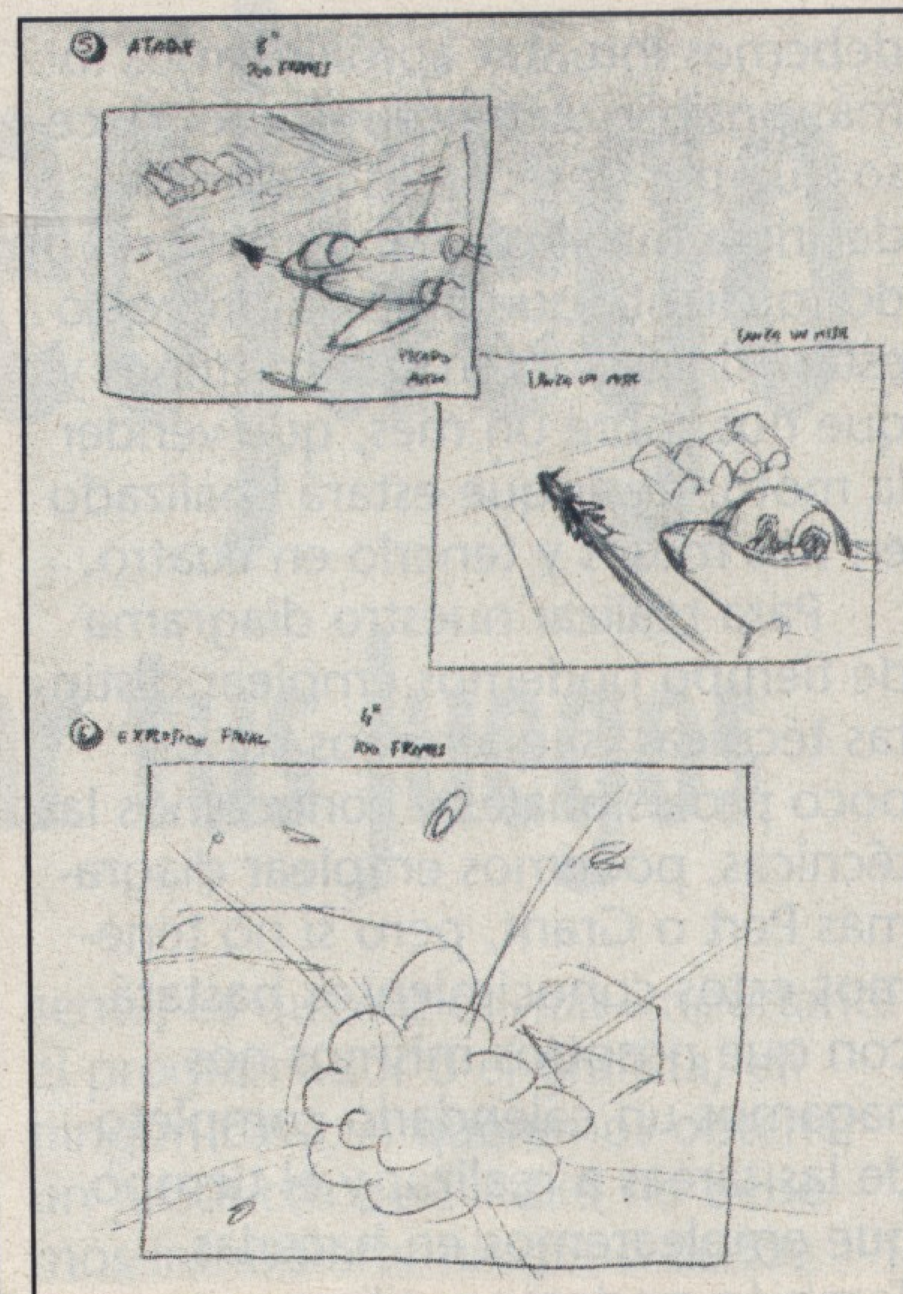
En el momento de estructurar el almacenamiento, todos los ficheros fuentes los tendremos que guardar como oro en paño, ya que cualquier modificación, sea del tipo que sea, nos será mucho más sencilla de realizar.

Antes de empezar con cualquier tipo de gráficos debe existir un guión o una idea fijada. Una vez conseguido lo anterior, se comenzará a realizar todo el trabajo con el material gráfico. El primer paso que se realiza para empezar el juego, en base al guión previamente establecido, es realizar un estudio mediante bocetos de la imagen que va a tomar el proyecto. Este primer paso es muy importante, ya que se trata del pilar de los gráficos, una vez que se tiene definida la línea de arte a seguir, continuamos con las primeras pruebas gráficas. Estas primeras pruebas deben basarse en trasladar los bocetos, que normalmente se encuentran en papel, a un soporte informático.

Dependiendo de la índole del juego se tomarán varios caminos, si el juego va a ser en 2D, sin ningún elemento en 3D, tomaremos los programas 3D como por ejemplo: *3DStudio Max, Lightwave, Maya, Softimage...*, únicamente como generadores de imágenes. Siempre que se opte por este sistema de creación de gráficos, también podemos utilizar programas de retoque fotográficos, principalmente el más utilizado es *Photoshop*.

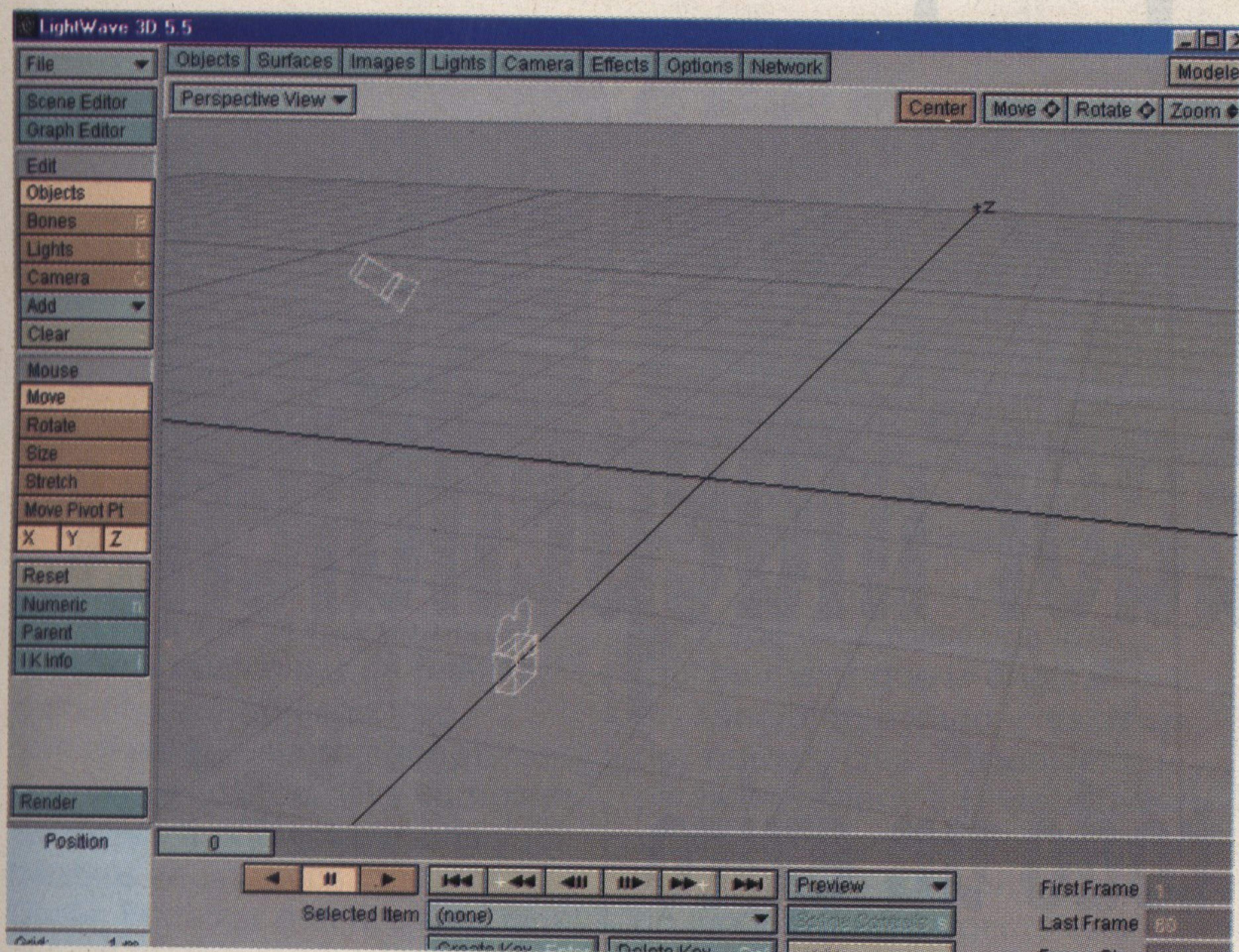
Si nuestro proyecto va a ser un videojuego 3D, debemos de incluir ficheros gráficos de geometría y de textura o imágenes "bitmap". También se deberán incluir todo tipo de animaciones. La obtención de todos estos datos se realizará mediante una exportación especial de nuestro programa 3D al motor 3D del juego. Si nuestro juego va a tener una intro o presentación crearemos un "story board" con todos los planos clave para una posterior representación.


Secuencias de imágenes para la elaboración de una intro mediante un story board.

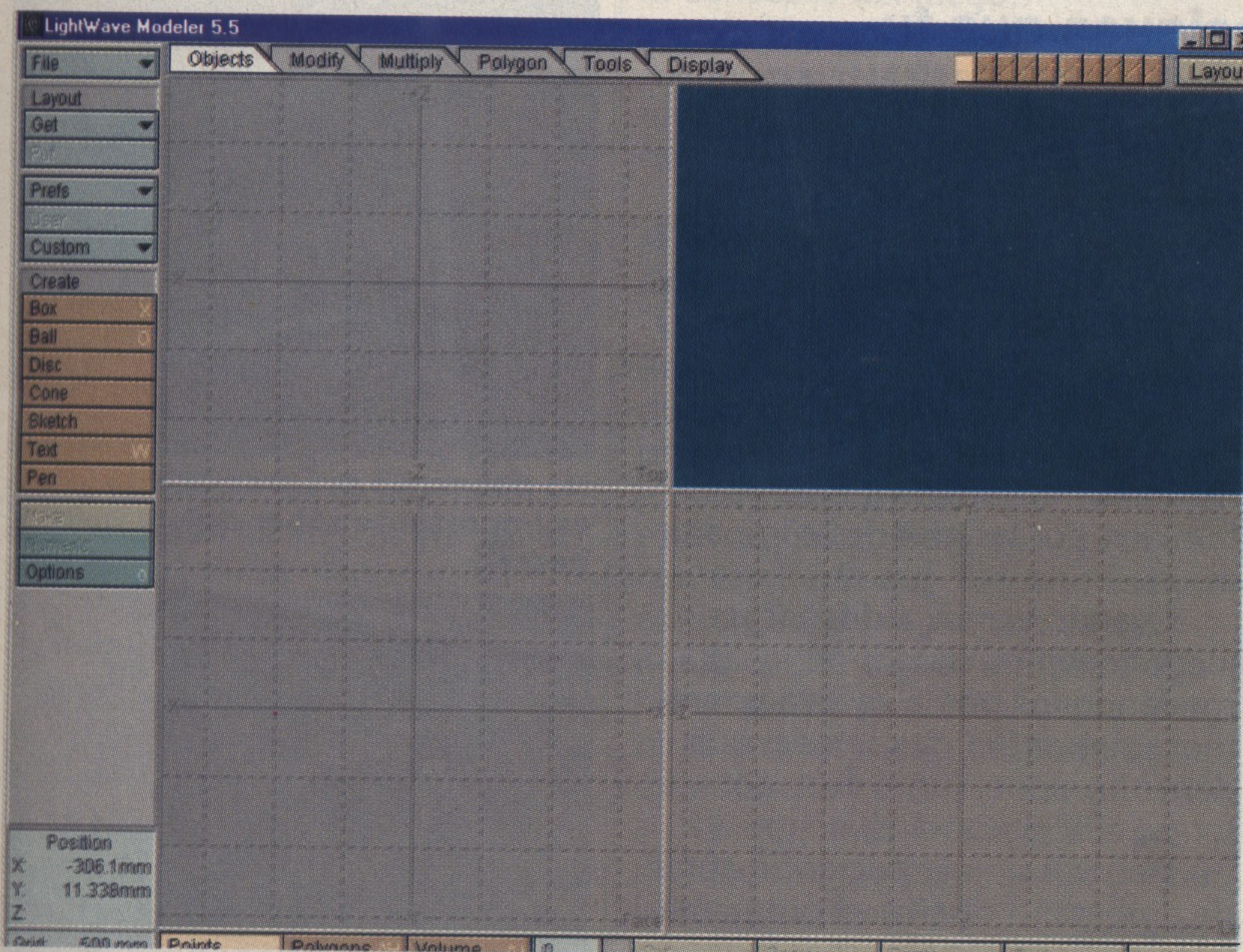

Bocetos de los diferentes personajes de nuestro juego.



Programa 3D Studio Max 2.5.



Programa Lightwave animate.



Lightwave modeler.

Creación de un story-board a modo intro

La creación del "story-board" es una parte muy importante del futuro videojuego. En la realización de esta tarea no sirve solamente que la persona o personas encargadas tengan unos conocimientos avanzados de dibujar, hay que representarlos de una forma clara y concisa.

En un "story-board" se plasman en imagen el guión del juego para meter al jugador en situación. Los más importante a tener en consideración es el tiempo que se le designa a cada toma, esto por una cuestión muy sencilla, no es lo mismo tener una toma de cuatro segundos, en la que se describe la acción de una persona caminando sola, a tener una toma de cuatro segundos de un grupo de personas que se encuentra jugando en la playa. La primera toma posiblemente se nos hará interminable, ya que por muchos detalles que se le quieran dar, buena iluminación y demás adornos, carece de información, de esta forma la toma será muy lenta y una perdida de tiempo en la animación. En cambio, la segunda toma es todo lo contrario, posiblemente no seamos capaces de asimilar toda la gran cantidad de información que produce el grupo de personas que se encuentran jugando en la playa.

Otro factor a tener en cuenta, si queremos captar la atención de las personas que estén viendo la intro, es no darles tiempo a que capten toda la información. Puede afectar al entendimiento del contenido, pero muchas veces con este método se consigue dar ritmo a la presentación, lo cual se incrementa con multitud de cambios de plano. Así se consigue una presentación vertiginosa que enganchará al espectador al juego desde el principio del mismo.

Todas las tomas deben de ir acompañadas de una estimación de tiempos. Estos tiempos se ajustarán a la acción que se quiere representar. En este apartado se debe poner gran atención si se quiere dejar un tiempo destinado a las transiciones de toma a toma. Si la acción se desarrolla en un período de 12 segundos, a este período de tiempo le añadiremos un tiempo extra destinado a la transición entre escenas.

Otro factor muy importante es la continuidad de las escenas. Obteniendo la adecuada continuidad en las sucesivas escenas conseguiremos que es espectador comprenda cuál va ser la ambientación del juego.

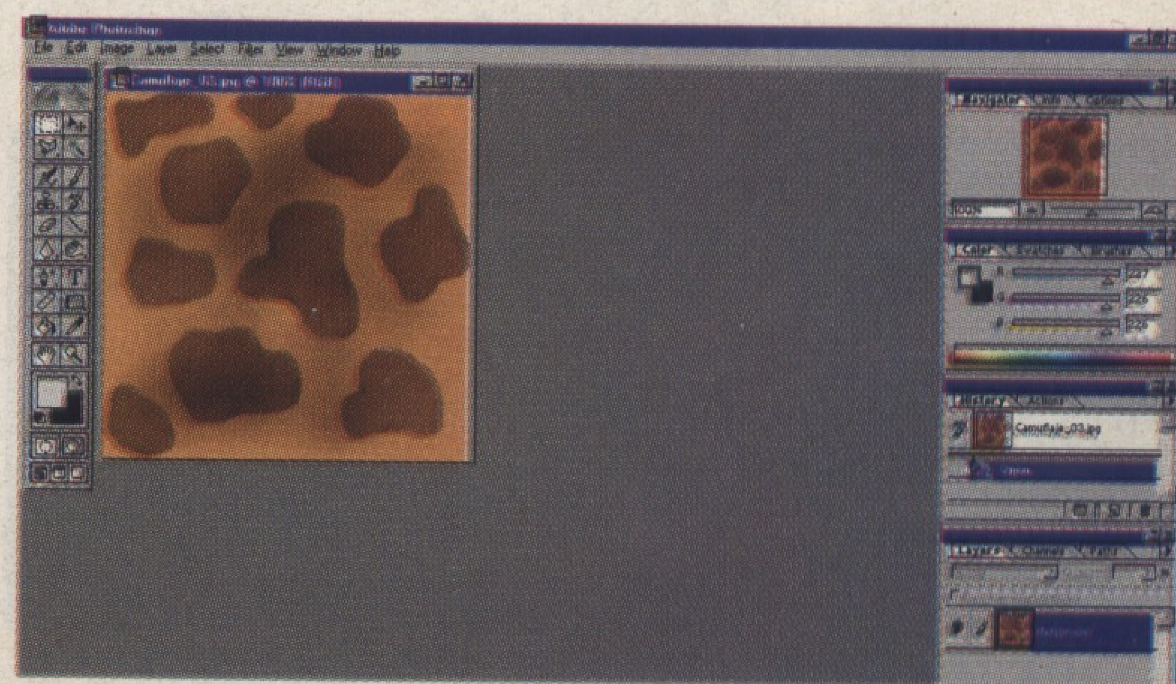
La historia o argumento de la intro debe de ser sencilla y además sutil, me refiero a que en un período tan corto de tiempo la información aportada debe de ser recibida y a su vez comprendida.

Para terminar el "story board" debemos acompañar unas pequeñas especificaciones de texto en la escena. Estas anotaciones tienen que contener datos que describan brevemente la acción de los diferentes elementos que intervienen. También pueden ir representados de una forma gráfica (una flecha, un arco, una línea...).

La introducción del juego debe ser lo más lujosa posible para meter al jugador en ambiente rápidamente

Con estas pequeñas nociones ya podemos empezar a planificar en que será nuestro próximo trabajo destinado a saborear las mieles del éxito. Sólo hace falta imaginación y los conocimientos técnicos necesarios para ponernos a realizar un grandioso videojuego, y mucho trabajo, por supuesto.

Armando Vélez y Raúl Bravo



El programa de retoque fotográfico Photoshop 5.0.

Los Diez Mandamientos del programador independiente

O cómo programar todo un éxito

Tras mucho esfuerzo consigues que tu protagonista cruce la pantalla. POR FIN!,-un frío pensamiento cruza tu mente-, ¿ya puedo programar un *Quake*? No todo es tan fácil como parece: la programación de juegos requiere conocimientos de programación, lógica, creatividad y sobretodo paciencia.

Ser un programador independiente no tiene muchas facilidades. Si analizamos las diferencias entre una compañía y cualquiera de los diseñadores que pueden leer esta revista seguramente llegaremos a una fácil conclusión: el dinero y la cantidad de gente disponible.

Las grandes compañías tienen una base económica en la que apoyarse para adquirir programas que facilitan enormemente la tarea de diseñar un juego. ¿Cuándo no hemos leído algún artículo de compañías que usan capturadoras de movimiento para crear los gráficos o programas que casi te hacen las imágenes ellos mismos?

Sí, hay muchas diferencias entre una compañía y un pequeño grupo de programación independiente, pero no debéis desanimaros ya que todas las compañías han empezado siendo pequeñas. Vosotros también podéis hacer un producto de calidad, y para empezar solamente tenéis que tener en cuenta los 10 mandamientos del programador independiente.

1er Mandamiento: Escribirás el diseño

Realmente es uno de los mandamientos más importantes y el primero a realizar no sólo por programadores independientes , sino también por las grandes compañías.

Cuando vamos a diseñar un juego tenemos cientos de ideas rondándonos por la cabeza y tenemos un montón de material que debemos poner junto para dar forma a lo que será el juego.

Debemos tener paciencia antes de empezar a diseñar el programa y los gráficos. Hay muchas maneras de prepararse para empezar el programa , pero quizás la más utilizada es la de escribir un *Documento de Diseño.* Un documento de diseño es un guión para tu juego realizado antes de la producción de éste. Los más básicos incluyen un resumen de la línea argumental del juego, los gráficos a realizar en general, así como una idea base sobre el control del juego. Algunos más complicados incluyen ideas como quién es el destinatario del juego, fechas límite para el diseño de los aspectos del juego o quién se encarga de realizar cada parte del juego.

Realizar un buen documento de diseño antes de empezar la programación nos ahorrará mucho tiempo durante la producción del juego, ya que permitirá que tengamos las ideas claras mientras vayamos creando nuestro juego.



No ponerse a programar a lo loco, es importante saber qué se quiere hacer.



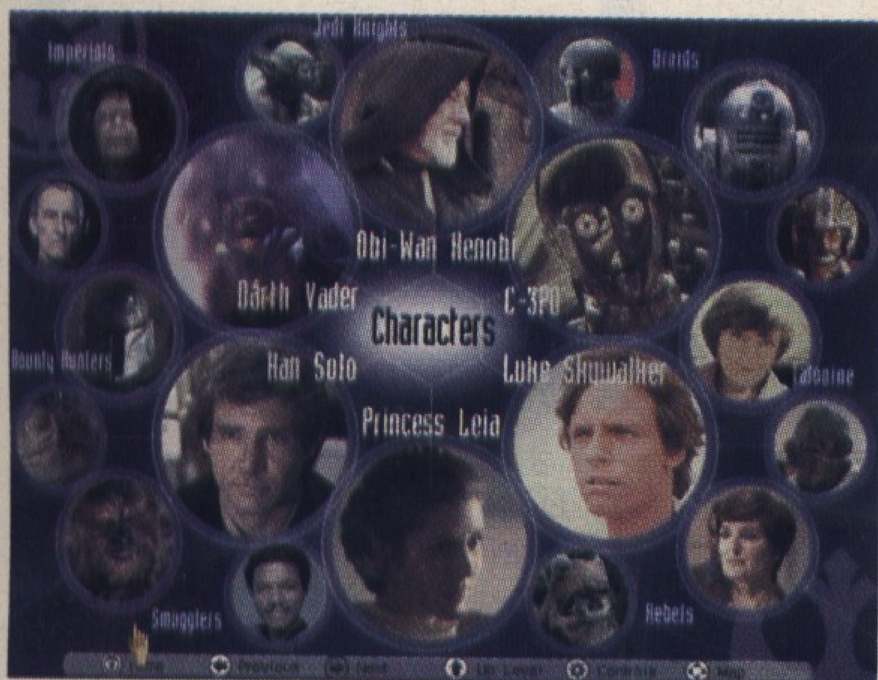
Al final podrás disfrutar de un trabajo bien hecho.

2º Mandamiento: Encajarás bien los conceptos

Mezclar géneros o dar con nuevas ideas puede realmente estar muy bien y puede acabar con la creación de magníficos juegos. Pero debemos ir con cuidado, no todas las ideas encajan perfectamente y puede ser que el resultado sea algo caótico e incomprensible para los jugadores.

No es malo copiar ideas de otros juegos que hayan tenido éxito o que nos gusten a nosotros particularmente como jugadores. Pero una cosa es eso y la otra crear engendros del caos sin sentido ni lógica: Creemos un juego tipo





El desarrollo de los personajes añadirá profundidad al juego.


La chispa es lo que hace que un juego triunfe o quede en el olvido.

Zelda, pero que al mismo tiempo tengamos que cuidar de nuestra ciudad en un subjuego tipo *SimCity* porqué el protagonista es el alcalde y cuando haya combates pasaremos a primera persona y los haremos en un escenario tipo *Quake*. Esta idea no encaja de ninguna manera y el jugador realmente perderá todas las ganas de jugar simplemente por la dificultad de aprender como funciona el juego y tener en cuenta la cantidad de información que va a tener que controlar durante todo el juego.

¿Por qué se han ido repitiendo el mismo tipo de ideas durante todas las épocas y se han creado los géneros? Es fácil, normalmente los jugadores escogen un juego por su finalidad: un arcade para descargar adrenalina, un wargame para desarrollar la mente y una aventura para desarrollar la imaginación. Mezclar los géneros puede estar bien, pero debemos pensar qué tipo de producto va a caer en manos del consumidor final y si va a ser agradable para el tipo de destinatario que tenemos pensado.

La finalidad de todo juego es que el que lo vaya a utilizar se lo pase bien, no que tenga dolor de cabeza.

3er Mandamiento: Dejarás que las ideas maduren

Las ideas son como el vino, con el tiempo mejoran y se hacen de mayor calidad. Antes de ponerte a programar el juego que se te acaba de ocurrir es mejor dejar pasar un poco de tiempo para que la idea vaya tomando forma progresivamente. Si la idea es buena debes tener por seguro que tu cerebro la perfeccionará casi sin darte cuenta de ello.

Algo que es fácil de hacer y da muy buenos resultado es comentar tu idea con personas de confianza que te hagan ver los pros y contras que tú no has visto por ti mismo, varias cabezas piensan más que una sola y seguro que entre todos se puede desarrollar una idea mucho mejor que la original.

Tomate tu tiempo, crear mundos nuevos, crear personajes que parezcan reales y a la vez interesantes, toma su tiempo. ¿Cuántos años pasa una compañía creando un juego? ¿Cuánta gente está dedicada únicamente a desarrollar el concepto del juego?

No hay prisa, eres un programador independiente y nadie te va a despedir porqué no tengas hecho un juego al año.

4º Mandamiento: Harás que tu juego resulte 'Guay'

Realmente el éxito de un juego en qué se basa: ¿unos gráficos sorprendentes? ¿una historia adictiva y novedosa?... Realmente todo eso está muy bien, pero no solamente con eso harás que tu juego sea un éxito. Cuando le preguntas a la gente que le parece un juego que ha tenido éxito normalmente recibirás una respuesta parecida a esta: 'Es Guay'.

Lo que debes hacer para que un juego tenga éxito es hacer que la idea sea interesante, adictiva, y sobretodo que al jugar te haga sentir algún tipo de emoción que no sientes normalmente.

Parémonos a pensar un momento en el Tetris, por poner un ejemplo. Coged a una persona que no haya oído hablar de ese juego y explicadle en que se basa: son unos bloques que tú puedes girar y dejar caer para hacer líneas que desaparecen al completarse. Realmente dicho de esta manera el juego puede parecer algo realmente muy aburrido. ¿Cómo es que el Tetris ha sido un éxito total y todavía se hacen versiones de este? Fácil: es un juego 'Guay': te ofrece un desafío a tus reflejos, sientes como la adrenalina te llena el cuerpo mientras los bloques caen cada vez más deprisa y sientes el triunfo cada vez que completas una línea.

Tu juego debe ser 'Guay', debe hacer sentir al jugador que ha tenido una experiencia inolvidable, que ha tenido un desafío nuevo y que se ha creado una relación entre él y el juego: tu juego debe ser **Adictivo**.

5º Mandamiento: Experimentarás hasta que salga bien

Las buenas ideas no acostumbran a ser el fruto de algo que acaba de cruzarse por tu mente. Durante la programación de tu juego deberías intentar probar distintas maneras de llevar a cabo la idea que tienes en mente, experimentando diferentes métodos y porqué no, creando métodos nuevos.

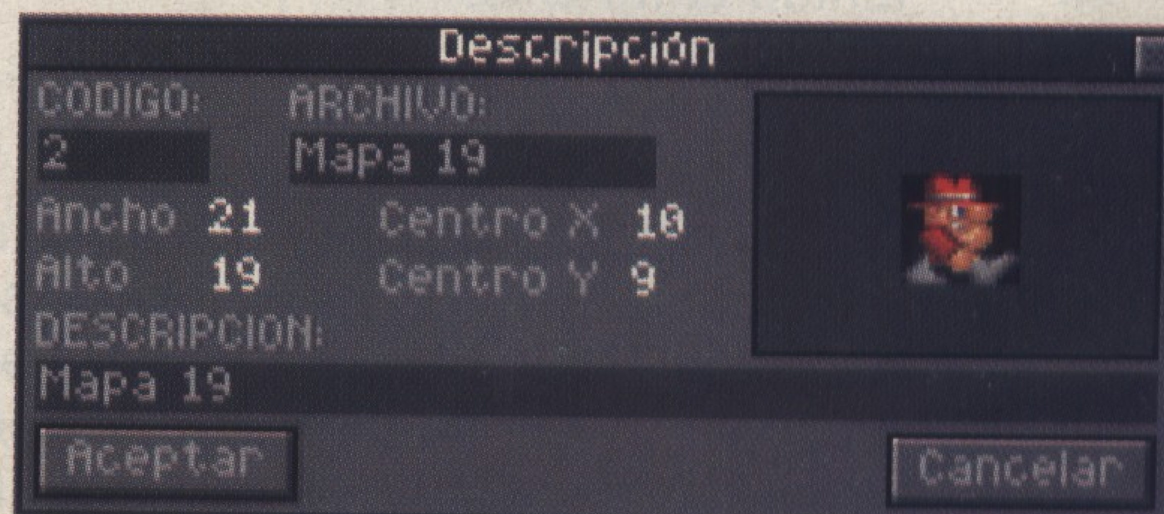
La utilización de ideas alternativas sólo está limitada a tu imaginación: sin plazos que cumplir y siendo tú y unos pocos los creadores, no tenéis que ajustaros a ningún tipo de reglas como deben hacer las grandes compañías. Se creativo y experimenta hasta que creas que lo que has hecho tiene una calidad excepcional.

Recuerda esto: antes del tetris no había ningún otro tetris, alguien creó la idea por primera vez. Se creativo y desarrolla ideas originales y experimenta con lo que te pase por la mente por absurdo que sea. Si no te gusta el resultado siempre puedes volver a empezar.

Las ideas son como el vino, con el tiempo maduran y mejoran

6º Mandamiento: Cuidar la jugabilidad

La jugabilidad: quizás de los primeros puntos que se deberían tener en cuenta en un juego. Un jugador que deba pasarse 2 horas leyendo el manual, 3 horas delante del ordenador con los apuntes al lado y varias consultas a unas FAQs (Preguntas frecuentemente realizadas) no es un buen juego. Debemos intentar que el jugador pueda divertirse lo antes posible sin tener que romperse el 'coco' intentado comprender una serie de instrucciones ininteligibles. ¿Por qué vamos a utilizar teclado, ratón y joystick si podemos hacer lo mismo con tres botones y un par de combinaciones?



Es necesario que las ideas maduren y alcancen cierta coherencia a medida que desarrollamos el programa.



Daikatana, un claro ejemplo de poca organización. Ha tardado casi 3 años en completar su desarrollo.

Algunas veces buscaréis el realismo como en los simuladores, con lo cual se intentarán crear motores complejos para recrear la dificultad de pilotar un submarino, un avión,... Pero en la mayoría de los casos, lo que buscamos es la diversión directa. Si puedes hacer que algo sea inteligible en 5 minutos, ¿ para qué hacerlo más difícil?.

7° Mandamiento: Harás juegos con 'chispa'

Parecerá de locos, pero ¿qué es lo que realmente nos atrae de los juegos?. Miremos un par de ejemplos claros: *Carmaggedon* y *Mortal Kombat*. Ambos han sido populares por un mismo motivo: la sangre. El hecho de atropellar peatones en el primero y el de acabar con tus contrincantes en el segundo ha atraído a miles de fans con algo novedoso y que hacen que el juego tenga algo especial, que tenga 'chispa'.

Publicar un juego y que lo acepten es una tarea realmente complicada

Quizás sea el 'lado oscuro' que tenemos en todos nosotros, pero mientras más morboso sea un juego, más nos atraerá. Si creamos juegos que tengan algo especial, que haga que juguemos una y otra vez para poder probar todas las posibilidades o creamos un juego en qué podamos sentir emociones que normalmente no sentiríamos, habremos conseguido algo muy importante para que nuestro juego tenga éxito: pensad sino porqué juegos como el *Resident Evil* han tenido tanto éxito.

8° Mandamiento: Racionarás las emociones

Muchos piensan que es mejor ganar en 20 minutos que perder en 10 y que es más divertido ganar en 2 horas que ganar en 20 minutos. Racionar la dificultad y la duración del juego hará que el que juegue se sienta satisfecho de lo que ha adquirido y de lo que le ha costado conseguir finalizar el juego que tiene entre manos.
Como las historias, todo juego debe tener una introducción, un nudo y un desenlace. Estas tres partes las debemos haber desarrollado durante la preproducción:

- La introducción: la primera parte de todas, frecuentemente un tutorial, debe permitir que el jugador aprenda el funcionamiento del juego y se empiece a familiarizar con lo que le espera más adelante. No debemos enseñar de buenas a primeras todo el potencial del juego, pero sí debemos dejar entrever lo que se va a encontrar más adelante.
- El nudo: debe ser la parte más larga del juego. Debe empezar de una forma pausada para ir aumentando progresivamente preparando al jugador para lo que será el 'Boum' final. Durante esta parte debemos mostrar poco a poco todo el potencial del juego y quizás dejar alguna pequeña 'sorpresa' para el final del juego.
- El Final: debe ser la síntesis de todo lo que le ha precedido. Solamente lo mejor del juego debe estar presente y debe ser lo que tenga mayor calidad del juego. Un buen final dejará satisfecho al jugador, un final sin calidad dejará al jugador con un mal sabor de boca: ¿para esto he estado perdiendo el tiempo delante del ordenador? Quizás se lo pensará dos veces antes de volver a adquirir uno de nuestros juegos.

9° Mandamiento: Saborearás la recompensa

El más importante de todos los mandamientos. Saborea el triunfo que da ver tu juego acabado, no te conformes únicamente con mostrarlo a los demás, sino que tómate la molestia de convertirte tú también en uno de los afortunados jugadores que probarán el juego.

Publicar un juego y que este sea aceptado es difícil, y ser rechazado por las compañías o por parte del público no debe deprimirnos. Sabemos que el juego que tenemos delante nuestro ha costado muchos meses de esfuerzos y más de un quebradero de cabeza. Sabemos que no ha sido fácil acabar lo que tienes entre manos, pero también sabemos lo que nos gusta a nosotros. Sabiendo esto, ¿para qué preocuparnos de lo que puedan decir si nos gusta el resultado final?

10° Mandamiento: Hallarás las respuesta en ti mismo

No todas las preguntas pueden responderse en los libros, en las revistas o en los artículos. La mayor parte de las dudas que puedas tener, las puedes responder tú mismo simplemente parándote a pensar un poco.

Piensa que tú eres el creador y que la única limitación que tienes es la que tú mismo te puedas poner. Existiendo los métodos, solamente hace falta que tú los pongas en práctica.

Por último sólo recordad esto, de cada 1000 juegos, uno sólo se convierte en un gran éxito. Si programáis uno, tendréis una oportunidad, pero si persistís tendréis muchas más oportunidades.

Jordi Martín



Con unas sencillas directrices y muchas horas de programación serás capaz de hacer juegos realmente buenos.

Introducción al diseño gráfico

Más gráficos para el juego de naves

Este artículo es una continuación del anterior que recordando era sobre un juego de naves. Este juego aportará al lector unos conocimientos ya bastante complejos de 3DS MAX necesarios para cualquier tipo de juegos. En este articulo también se va a crear una presentación para el juego.

En el artículo anterior se realizó un meteorito de una manera muy rápida por lo que en este número se va a empezar por describir más extensamente la manera de diseñar los meteoritos. Los pasos a seguir son los siguientes:

- Primero de todo se debe crear una esfera del tamaño que más o menos deseemos para el meteorito.
- A continuación se debe ir al panel de modificadores tal y como muestra la figura 1. (este panel ya fue utilizado a la hora de localizar el modificador *extrude* o *lathe*).
- Seleccionar el modificador de *Noise*.

<u>Nota</u>: Es posible que el modificador *noise* no esté disponible entre los botones *estándar* del panel modificadores. Para encontrarlo lo que se debe hacer es ir al botón que pone *more* y buscar el modificador en la lista de modificadores.

El modificador *Noise* es un modificador que realiza pequeños cambios en la malla poligonal del objeto de manera que parece que está más arrugado. Este modificador es especialmente útil si se quiere representar imperfecciones en un objeto. Por lo general, un objeto tridimensional suele ser siempre bastante perfecto así que este modificador es la herramienta perfecta a la hora de crear imperfecciones aleatorias en la malla poligonal. Este modificador tiene bastantes parámetros entre los cuáles podemos definir cómo de grande queremos el ruido o cosas semejantes. Para poder usar cualquier modificador es preciso que haya algún objeto seleccionado. En nuestro caso, la esfera a la que queremos convertir en meteorito.

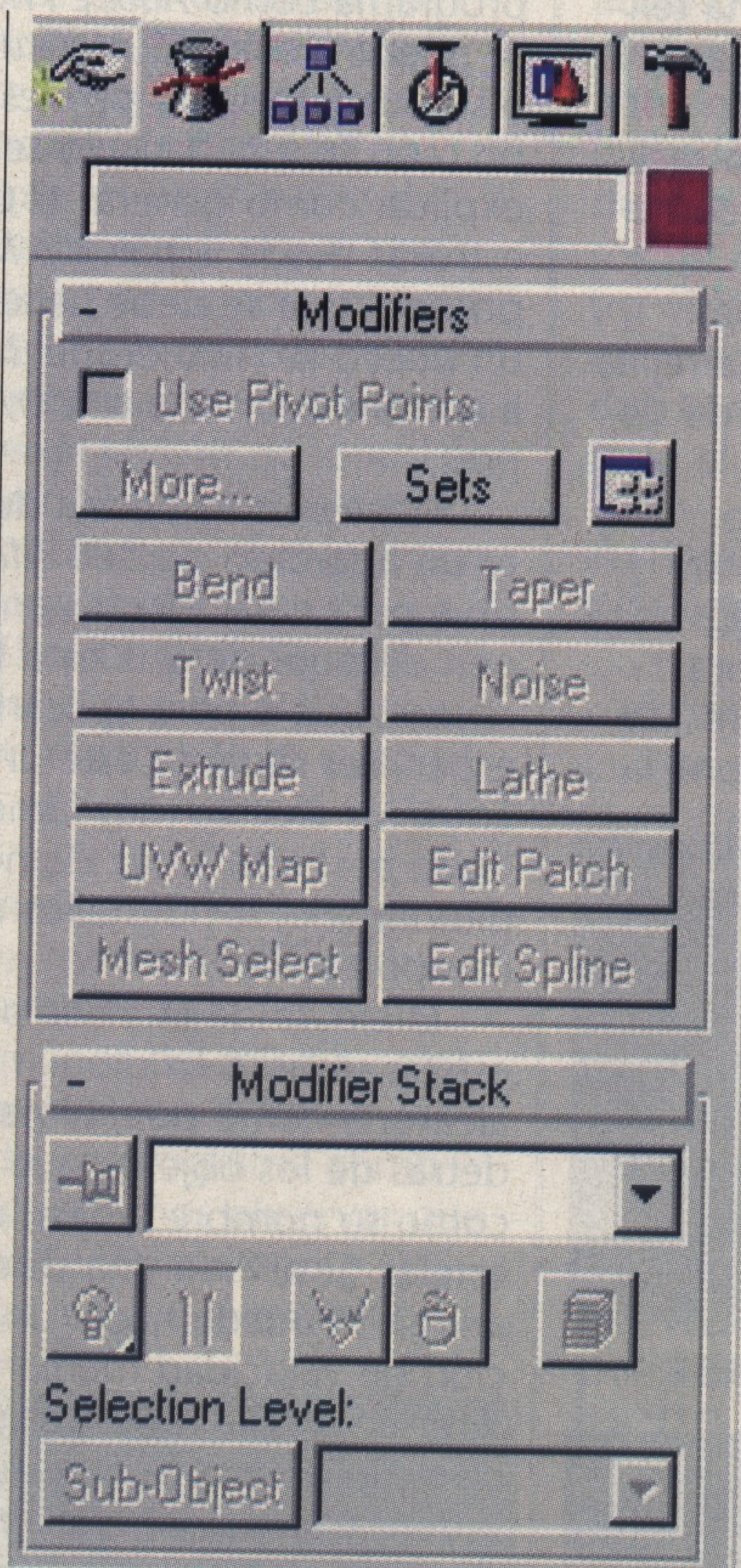


Figura 1 - Panel de modificadores.

- Para un resultado adecuado, el autor del articulo ha utilizado los valores que se muestran en la figura 2 (estos son aleatorios dependiendo del tamaño de la esfera y del efecto deseado), es decir, los valores usados son:

En el apartado *Noise* del modificador *noise*:
– Scale: 42,78

En el apartado *Strength*:
– X: 36,53
– Y: 22,91
– Z: 75,07

Se debe recordar que estos valores son variables y dependen del volumen de la esfera, numero de segmentos...

Una vez aplicado el modificador, el asteroide sin textura debería quedar como la figura 3.

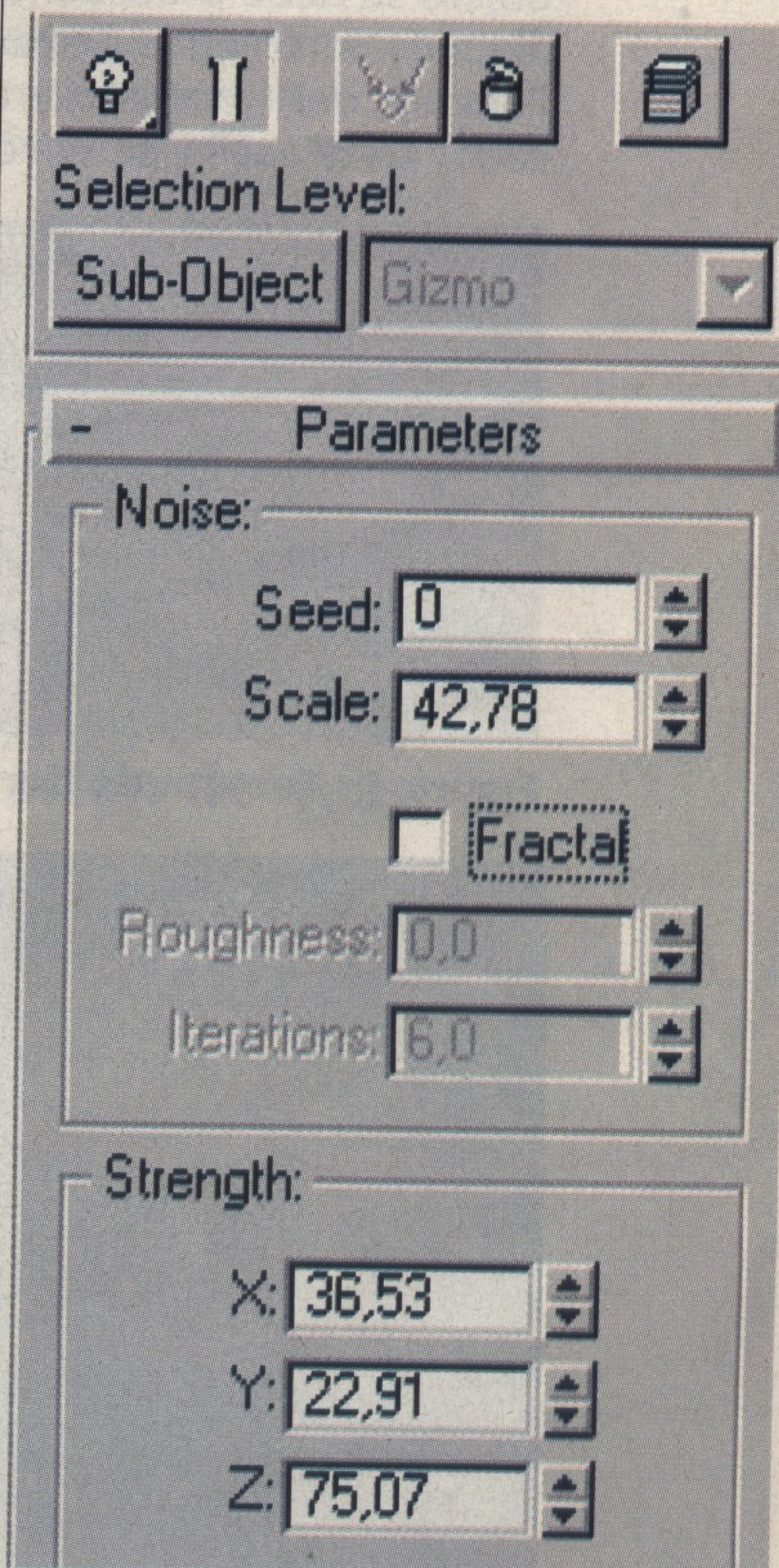


Figura 2 - Parámetros del modificador.



Figura 3 - meteorito sin textura.



Figura 6 - Campos estelares con *blur.*



Figura 7 - Campos estelares sin *blur.*

Una vez realizado este meteorito, es hora de darle textura y con ello, un poco más de vida. Para un meteorito una textura que podría servir seria una textura de tierra, (no la tierra como planeta sino la tierra que pisamos a diario). Para ello, tal y como se dijo en el articulo pasado se puede recurrir a sitios como www.3dcafe.com o semejantes. El autor ha utilizado una textura como la de la figura 4.

Para un resultado adecuado se han utilizado los valores de la fig. 2

Seguidamente los pasos a realizar para darle textura al meteorito son los siguientes:

- Entrar al editor de materiales.
- Crear un material que en la rejilla de *diffuse* tenga la textura de tierra.
- Aplicar el material al objeto.

Una vez aplicado el material al objeto se debe proceder a ajustar algunos parámetros de textura. Para ello debemos entrar al modificador *UVW MAP* y se selecciona la



Figura 4 - Textura de tierra.



Figura 5 - Render del meteorito con textura.

forma de *mapeado* que quede mejor para el objeto que tenemos, en este caso el meteorito. En el caso del autor se ha utilizado un *mapeado* cúbico que ha dado como resultado la figura 5 después del *render*.

El campo estelar

Existen varias formas de realizar un campo estelar. Se puede usar un programa como *Adobe Photoshop*, se puede crear con *3D Studio MAX*, o se pueden pintar píxeles en tiempo real. En este articulo se va a explicar como generar el campo de estrellas desde *3D Studio MAX*. Lo primero que se debe hacer es crear un nuevo archivo desde *3DS MAX* y a continuación ir a *render/video post*.

Video post se podría decir que es un apartado de *3DS MAX* que sirve para crear efectos en la imagen después de que esta haya sido *renderizada* o para aplicarle efectos especiales durante este proceso. Entre los efectos más famosos del *Video post* está la luz de neón, el difuminado, los destellos de luz y el campo estelar.

El efecto de *Starfield* o campo estelar se basa en dibujar pixeles aleatoriamente por la imagen, detrás de los objetos que simulan como su nombre indica, un campo estelar. Si se crea una cámara y se anima se puede hacer una simulación de velocidad mediante un difuminado de las estrellas. Un ejemplo de esto son las figuras 6 y 7 donde se puede observar claramente el campo de estrellas difuminado y sin difuminar.

Trabajando con Video Post

Video post es una herramienta muy potente y como se ha dicho antes se pueden hacer efectos en el *render*. Desde poner otra secuencia ya *renderizada* antes de la que se va a *renderizar* ahora, a añadir efectos espectaculares como la luz de neón o los campos de estrellas.

Los efectos como la luz de neón o los campos estelares son los llamados filtros de vídeo. Pero antes de poder aplicarlos debemos aplicar una imagen que se genera. Normalmente esta imagen será la que está generando el render.

Para hacer esto lo que se debe hacer es lo siguiente:

- Seleccionar el botón de *"Add scene Event"*. Con este botón lo que haremos es que la ventana que seleccionemos será por la que entre el vídeo. Estas suelen ser: *Top, left, perspective* entre otras... Si hemos creado una cámara que enfoque al meteorito, deberemos seleccionar la cámara en el *listbox*, de lo contrario deberemos seleccionar la vista *perspective*. Lo que estaremos haciendo de esta manera es que los objetos representados en la ventana que se ha seleccionado entrarán en *Video Post* para ser procesados más tarde. El botón de *"Add scene Event"* es como el de la figura 8.
- Ahora es el momento de añadir las estrellas que como hemos explicado antes se van a incluir en la escena con el filtro llamado *Starfield*. Para poder seleccionar dicho filtro lo que se debe hacer es pulsar en el botón de *"Add scene filter event"* que está justo dos botones a la derecha del de *"Add scene Event"*, y de entre los filtros, seleccionar el de *StarField*. Una vez seleccionado se podrá observar que se puede entrar a la configuración del filtro pulsando sobre el botón que pone *"Setup"* como muestra la figura 9.

Se puede ver que una vez dentro de la configuración del *Starfield* se obliga al usuario a seleccionar

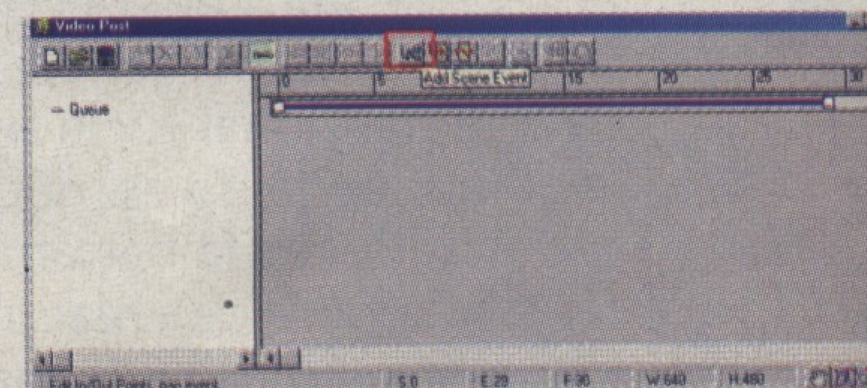


Figura 8 - seleccionando el *Add Scene Event.*

una cámara. Esta cámara es la que afectarán a las estrellas. ¡¡En efecto!! Es obligatorio tener una cámara creada. No se pueden hacer estrellas desde una vista perspectiva. En artículos anteriores ya se explicó la manera de crear las cámaras, así que no se va a volver a decir.

Una vez configurado el número de estrellas que queremos, y completar todos los parámetros necesarios es hora de hacer el *render*. Para ello, no seguiremos el mismo procedimiento que se seguía hasta ahora sino que se va a utilizar otra técnica diferente. De nada nos servirá configurar la resolución de la imagen ya que el *render* del *video post* es independiente del *3ds MAX*. Es decir, la ventana de los parámetros del *render* que se abría en anteriores artículos queda inutilizada teniendo que usar la del *"video post"* que es más sencilla y con menos parámetros.

Para poder *renderizar* una imagen lo único que debemos hacer es pulsar sobre el botón que tiene el dibujo de un hombre corriendo en el *video post*. Al pulsarlo deberá salir una ventana en la que podemos definir la resolución a la que se va a *renderizar* la imagen, también podremos definir si queremos *renderizar* solo un fotograma o por el contrario si queremos *renderizar* muchos fotogramas. En el caso de nuestras estrellas deberemos definir una resolución que sea igual a la que generará el juego de manera que encaje perfectamente en la pantalla. Es decir, si el juego de naves va a ser a 800x600 lo que deberemos hacer es poner esa misma resolución. También es importante advertir que sólo se debe hacer el *render* de un solo fotograma. En este caso se nos da a escoger qué fotograma es aquel que nosotros queremos.

Nota: Este *render* se debe hacer sin el meteorito pues no forma parte de las estrellas sino que es un elemento independiente que va avanzando por la pantalla.



Figura 10 - Meteorito con fondo verde.

Otros elementos a tener en cuenta

Hay algunos elementos que debemos tener en cuenta para hacer bien los gráficos de un juego como se comentó en los primeros artículos. Se debe destacar que el color de fondo de una imagen no es recomendable que sea negro ya que el negro es un color que suele aparecer en las sombras de los objetos. Esto puede provocar cosas como que en medio del juego salga un trozo de nave o meteorito transparente, cosa que no es demasiado recomendable y se puede considerar como una catástrofe. Para evitarlo, tenemos en MAX una forma de solucionarlo.

Lo que se debe hacer es entrar al apartado *environment* que está dentro del apartado *Rendering* de los menús de *3D Studio MAX*. Una vez realizado esto se abrirá una ventana con algunos parámetros importantes. Hay uno especial que además nos será muy útil para que no tengamos problemas con las transparencias. Este elemento importante es *color*. Junto a esta palabra podemos encontrar un recuadro que en principio es de color negro pero que si pulsamos sobre él podemos cambiar el color. Este color será el que se use para el fondo, de manera que si ponemos un color verde chillón de fondo es posible que no nos interfiera con ningún otro objeto y el color transparente sea perfecto.

Nota: El color seleccionado puede ser otro que no sea el verde chillón pero el autor lo ha usado porque no es un color que suela aparecer.

De manera que si volvemos a *renderizar* el meteorito, se podría observar como el de la figura 10.

Si pintáramos del mismo color verde la imagen con el fondo negro desde un programa como *Photoshop* podríamos observar que una gran parte del meteorito sería transparente, cosa que no interesaría. Esto se puede ver reflejado en la figura 11.



Figura 11 - Un error muy común.

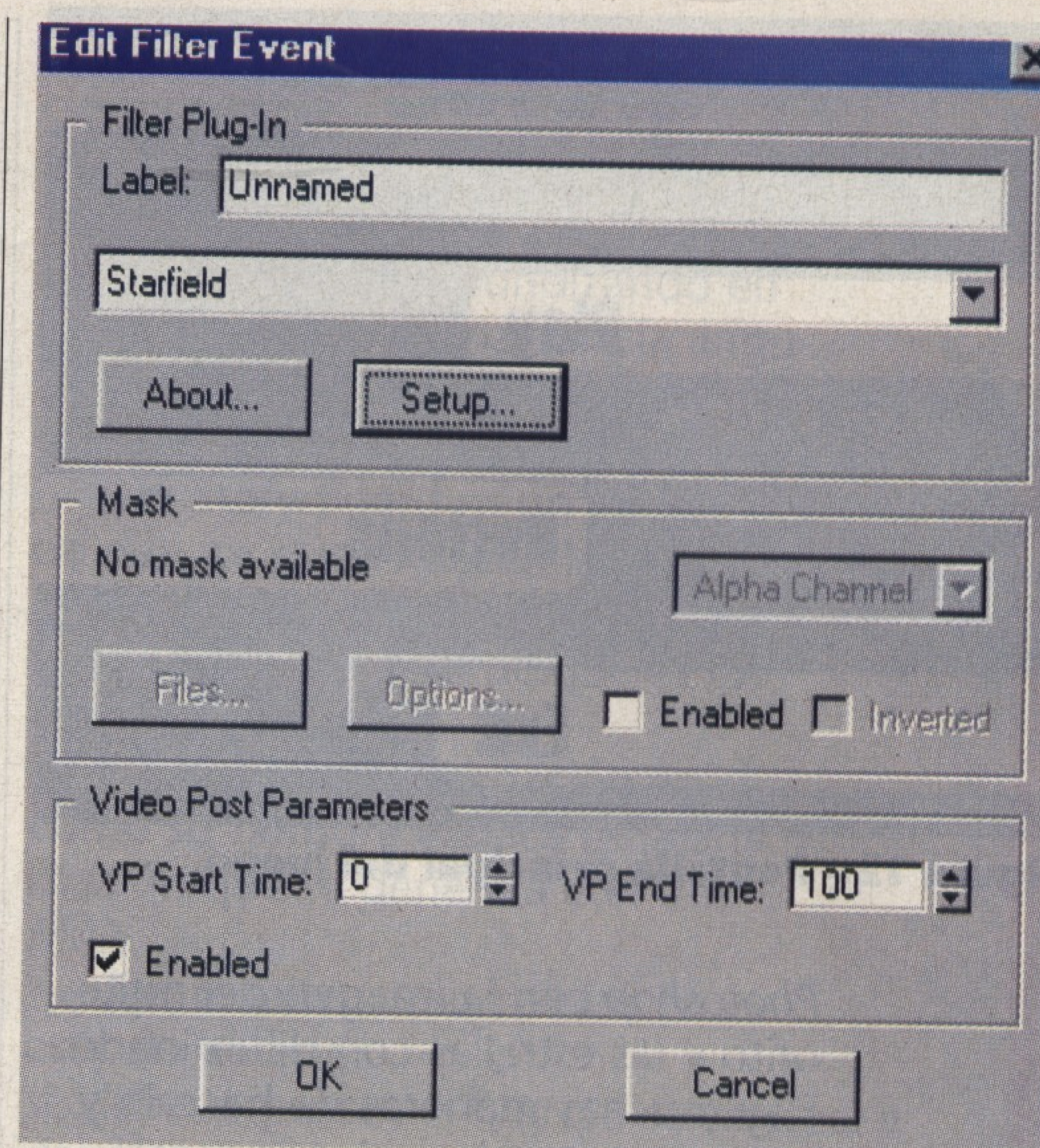


Figura 9 - Configuración de Starfield.

El color de fondo de una imagen no debe ser negro

Es por esta razón que el autor de este artículo recomienda usar la técnica anteriormente mencionada. Es importante también mantener en disco o cualquier soporte los ficheros de *3DS MAX* referentes a vuestro objeto de manera que si hay que hacer modificaciones se puedan hacer sin tener que rehacer todo el objeto.

Antes de escribir el código

Como es lógico, antes de escribir el código fuente del juego se debe pensar exactamente como va a ser el mismo. Qué métodos y trucos vamos a utilizar y cómo se puede optimizar más el programa. En el número anterior de la revista hubo un articulo totalmente dedicado a este tema (la optimización de programas).

Nota: El código del juego se va a escribir en el próximo artículo.

Pantallas de presentación de nuestro juego

Con programas como *Photoshop* o semejantes es posible hacer verdaderos milagros. A la hora de hacer las pantallas de presentación estos programas juegan un papel muy importante puesto que no tienen tridimensionalidad alguna. Como mucho puede ser la imagen de fondo en 3D pero es difícil encontrar una pantalla de presentación totalmente realizada en 3D.

Un ejemplo de pantalla principal podría ser algo parecido a la figura 12 donde se han fusionado efectos directamente creados con

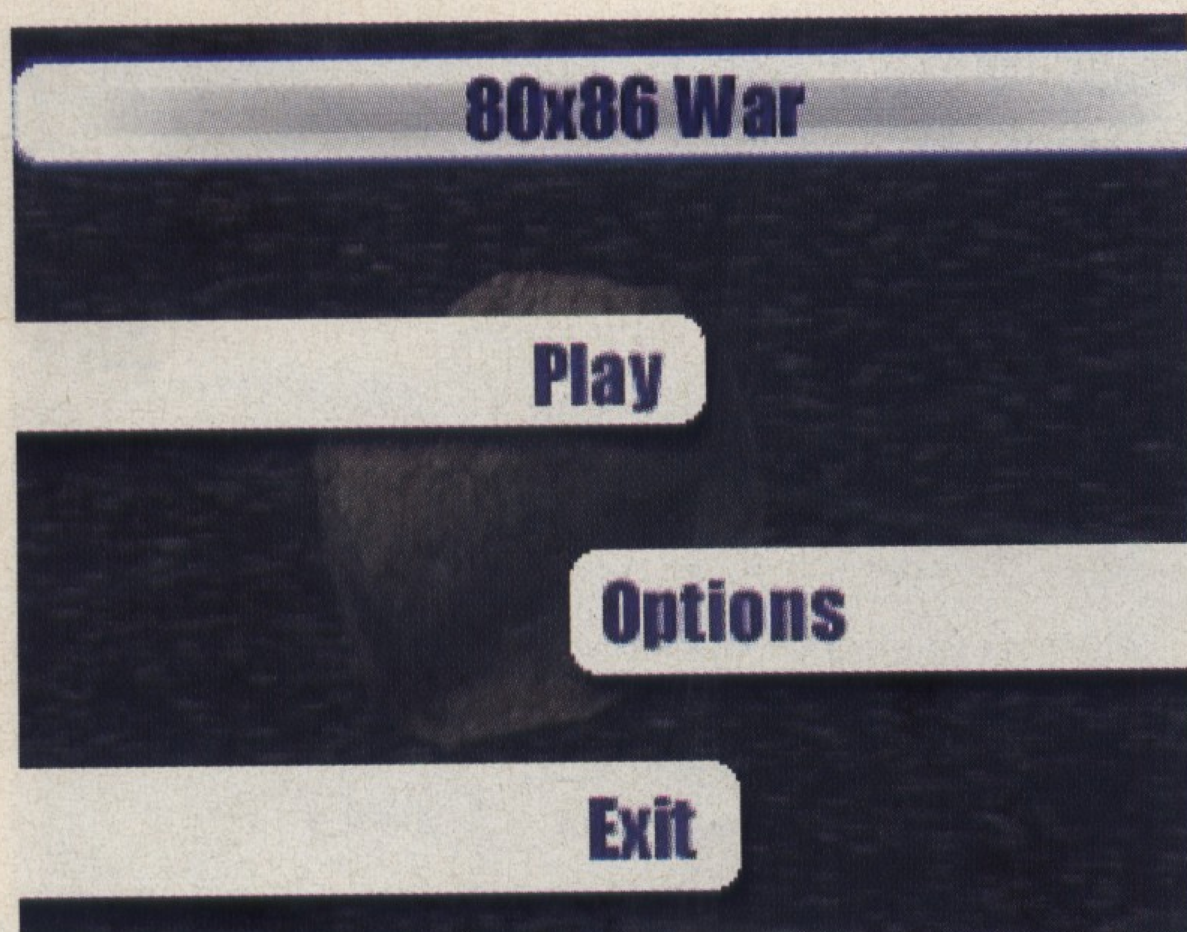


Figura 12 - Pantalla principal del juego.

Photoshop con la imagen del meteorito y las estrellas con difuminado. Hay muchas maneras de hacerlo y para encontrar la mejor se recomienda ir probando varias hasta quedarse con la que más nos gusta. Debemos recordar que DIV solo soporta 256 colores por lo que no deberemos pasarnos al hacer la imagen que, como ya se explicó en el articulo nº 1, debe ser de color indexado. Como se dijo también en el articulo 1, podemos dedicar toda una paleta a una sola pantalla pero a veces incluso con 256 colores no hay suficiente para poder representar aquello que queremos.

Con programas como Photoshop es posible hacer verdaderos milagros

La presentación del juego

A continuación se va a hacer una introducción a la animación mediante 3ds MAX y se va a empezar a hacer una presentación para el juego.

Para animar desde cualquier programa de 3D debemos tener en cuenta que trabajamos en un espacio de 3 coordenadas, X, Y, Z y que, en la mayoría de los programas, si tenemos un objeto en un sitio en el fotograma 0 y en el fotograma 100 lo movemos a otro sitio, este se moverá solo en el resto de los fotogramas.

Para animar en 3ds MAX se debe recordar mucho un botón que es, *Animate*, que está abajo y a la derecha de la pantalla cuando se arranca el MAX. Este botón es importante a la hora de animar. Incluso su nombre nos indica que sirve para ello. Si no se usa ese botón será imposible generar animación alguna.

Para realizar animaciones lo que se debe hacer es pulsar el botón *animate*, seguidamente se debe mover el objeto deseado a la nueva posición y desactivar el botón de *animate*. Si se prueba ahora a pasear por los diferentes fotogramas se podrá observar como el objeto se mueve solo. Esta es una gran ventaja respecto al mundo de la animación bidimensional puesto que en ella se debe dibujar fotograma a fotograma mientras que la animación 3D hace los cálculos por nosotros. Esto no quiere decir que la animación por ordenador sea fácil ni mucho menos. Lo que quiere decir es simplemente que los objetos se muevan, roten o se escalen, pero para que una cosa parezca mínimamente real se debe conocer bien el movimiento de tal objeto. Por poner un ejemplo, los movimientos de un robot futurista podrían ser difíciles de realizar de una forma convincente por no hablar de animar a una persona o a un animal como puede ser un dinosaurio. Se podría decir que la animación es una de las partes más complejas del 3D.

Un movimiento básico

Se va a generar un ejemplo en el que un cubo se mueva por la pantalla (nada especial) pero de suma importancia porque nos va a definir las bases para animar con *3ds MAX*.

Para empezar se debe crear un cubo, cosa que ya se sabe hacer, después se pulsa sobre el botón *animate* y colocamos el número de fotogramas a 100 de manera que estemos modificando el fotograma numero 100. Seguidamente pulsaremos el botón de mover o bien pulsaremos con el botón derecho sobre el cubo y escogeremos el comando *move*. Una vez pudiendo mover el objeto, se deberá arrastrar hasta otro punto del espacio tridimensional. Entonces se deberá desactivar el botón de *move* de manera que se pueda finalizar la animación.

Nota: No es necesario modificar el fotograma 100 (en este caso el último) sino que se puede ir realizando en todos los fotogramas deseados. Ya sea de 15 en 15, 4 en 4 o no tienen porqué seguir ningún tipo de orden específico.

Si ahora nos paseamos por los fotogramas podríamos observar que el cubo se va moviendo solo por el espacio tridimensional. Pues bien, esto es una animación de lo más básica. Ahora va siendo hora de añadirle algunas cosas para mejorarla.

El track View. Esta es una herramienta muy potente que nos permite animar de una forma más compleja que no es solo mover, escalar o rotar. Aquí tenemos algunos parámetros que harán que la animación tenga un toque más realista. Por ejemplo, cuando un reloj de agujas va marcando los segundos, el movimiento de la aguja es muy rápido, imperceptible para el ojo humano. Es inútil intentar representar eso moviendo cada fotograma. Más fácil es usar otras técnicas que pueden encontrarse en el *Track View*.

La potencia del *Track View* es a su vez lo que hace que esta sea una de las partes mas complicadas de todo el *3ds MAX*, aunque con él se pueden hacer las mas extraordinarias animaciones.

Para poder entrar en el *Track View* lo que se debe hacer es en el menú *Track View* darle a la opción *Open Track View*. Para el usuario nuevo en 3D puede pensar... "¡Cuantos parámetros!" Pero tranquilos, cuando se sabe un poco de que va la cosa, es más fácil de dominar. Al abrir el *Track View*, este, debería mostrar un aspecto más o menos como el de la figura 13.

Lo primero que se va a hacer es una animación básica, en la que el cubo desaparezca de un sitio y aparezca en otro. Lo que se debe hacer para ello es, una vez con la animación del cubo, abrir el *Track View*, una vez abierto se debe entrar al apartado de los objetos, una vez dentro se debe seleccionar el objeto que queremos modificar, en nuestro caso el cubo. Una vez seleccionado se deben abrir todos sus *sub-apartados*.

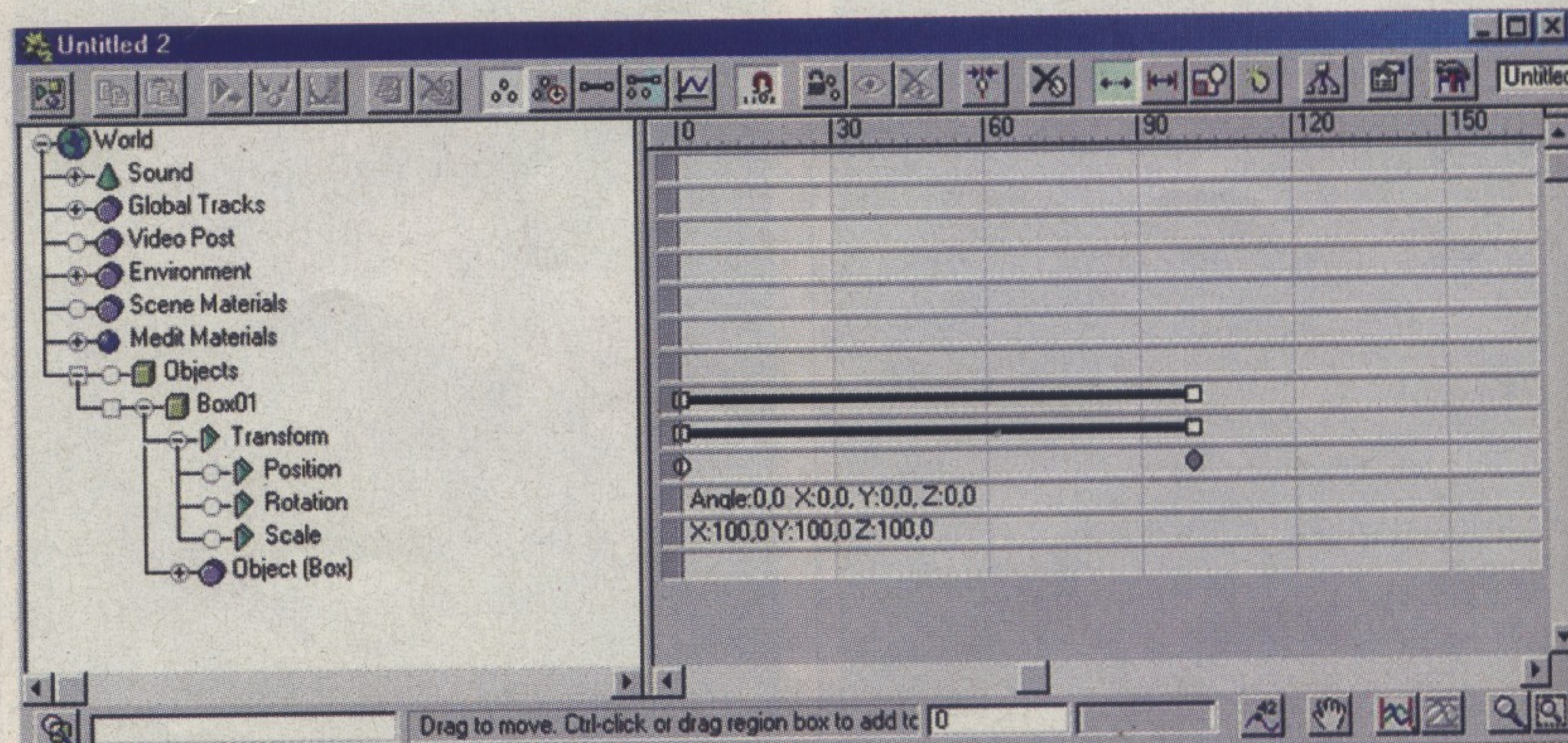


Figura 13 - Track View.



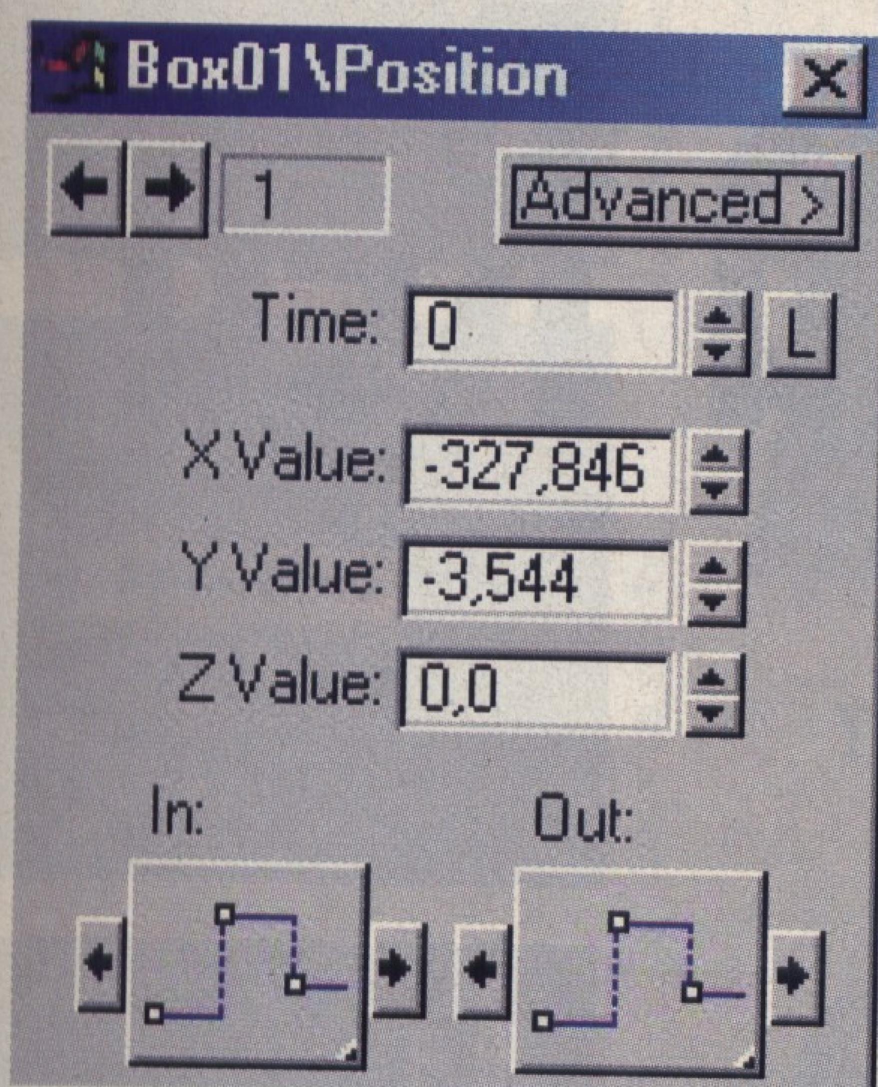


Figura 14 - Ventana de configuración.

Se podrá observar que en el apartado de *position* tiene dos puntos en caso de que hayamos modificado dos fotogramas o más puntos en caso contrario. Pues bien, si se pulsa con el botón derecho sobre uno de estos puntos nos saldrá una ventana como la de la figura 14.

Como puede observarse, en la parte baja de esta ventana hay dos dibujos que definen el estado del objeto en la animación. Pues bien, si se pulsa sobre uno de estos dos objetos se podrá cambiar. Para hacer la animación antes mencionada, se deberá pulsar sobre los dos dibujos y dejarlos como en la figura 13. Si ahora ponemos *play* en el reproductor de fotogramas de *3ds MAX* se podrá observar como, en un momento dado, el cubo desaparece de un sitio y aparece en otro.

Para poder observarlo mejor, se debe pulsar con el botón derecho del ratón sobre el botón de *play* del reproductor de fotogramas de *3ds MAX*. A continuación se debe cambiar el parámetro de *End Time* de 100 a 200 de manera que ahora la animación dura 200 fotogramas, es decir 100 fotogramas más que antes. Si ahora se ejecuta el *play* se podrá observar la animación sin problemas. Se puede ver un ejemplo de esa ventana en la figura 15.

Cambiar la duración de una animación. Con el *track View* también se pueden hacer cosas como cambiar la duración de la animación, para ello se abrirá el *track View* y se abrirá el apartado objetos, seguidamente se deberá abrir el apartado cubo y se podrá observar que a su derecha hay una línea que empieza en el fotograma 0 y acaba en el 100. Pues si se coge esta línea por donde acaba y la movemos hasta el fotograma 75 podremos observar que lo que antes hacía nuestro cubo en 100 fotogramas ahora lo hace en 75.

Generando la presentación

Ahora va siendo hora de ponerse a trabajar con la animación de presentación en 3D del juego. Para ello se deberá disponer de los objetos generados hasta ahora como la nave, los meteoritos (se pueden copiar), las naves de los "malos", y todo aquello que se quiera en la presentación del juego, así como lasers o lo que sea.

Antes de empezar a animar sin saber lo que se va a hacer se debe pensar cómo hacerlo, qué factores se deben tener en cuenta y cómo se puede realizar. Para ello, es bueno generar las siguientes cosas:

- Una sinopsis argumental.
- Un guión técnico.
- Un Story Board.
- La animación en sí.

En los próximos artículos se va a empezar a profundizar sobre cómo generar todos estos guiones y cosas importantes a la hora de crear cualquier película, anuncio publicitario, animaciones o casi cualquier cosa audiovisual. Pero de momento se va a hacer una breve introducción:

Una sinopsis argumental es como un escrito que explica todo lo que pasa en la animación, anuncio o película.

Un guión técnico explica secuencia a secuencia y plano a plano la acción, profundizando en cosas como la iluminación, la angulación, los movimientos de cámara etc...

Un Story Board es un conjunto de dibujos que explican la acción de manera visual para que se tenga una idea más global de qué es lo que se desea hacer.

Finalmente la animación en sí, como su nombre explica es la animación que se generará con *3ds MAX*.

Ahora tal y como se hizo en el articulo anterior se debería proceder al apartado de dudas de los lectores pero parece que ningún lector tiene preguntas del articulo pasado. Aunque esto sea así, se recuerda que cualquier duda o imágenes generadas por los lectores pueden ser enviadas a dmc@mundomail.net para su publicación en este apartado de la revista.

> Antes de empezar a animar hay que tener en cuenta varios factores

David Martínez (dmc@mundomail.net)

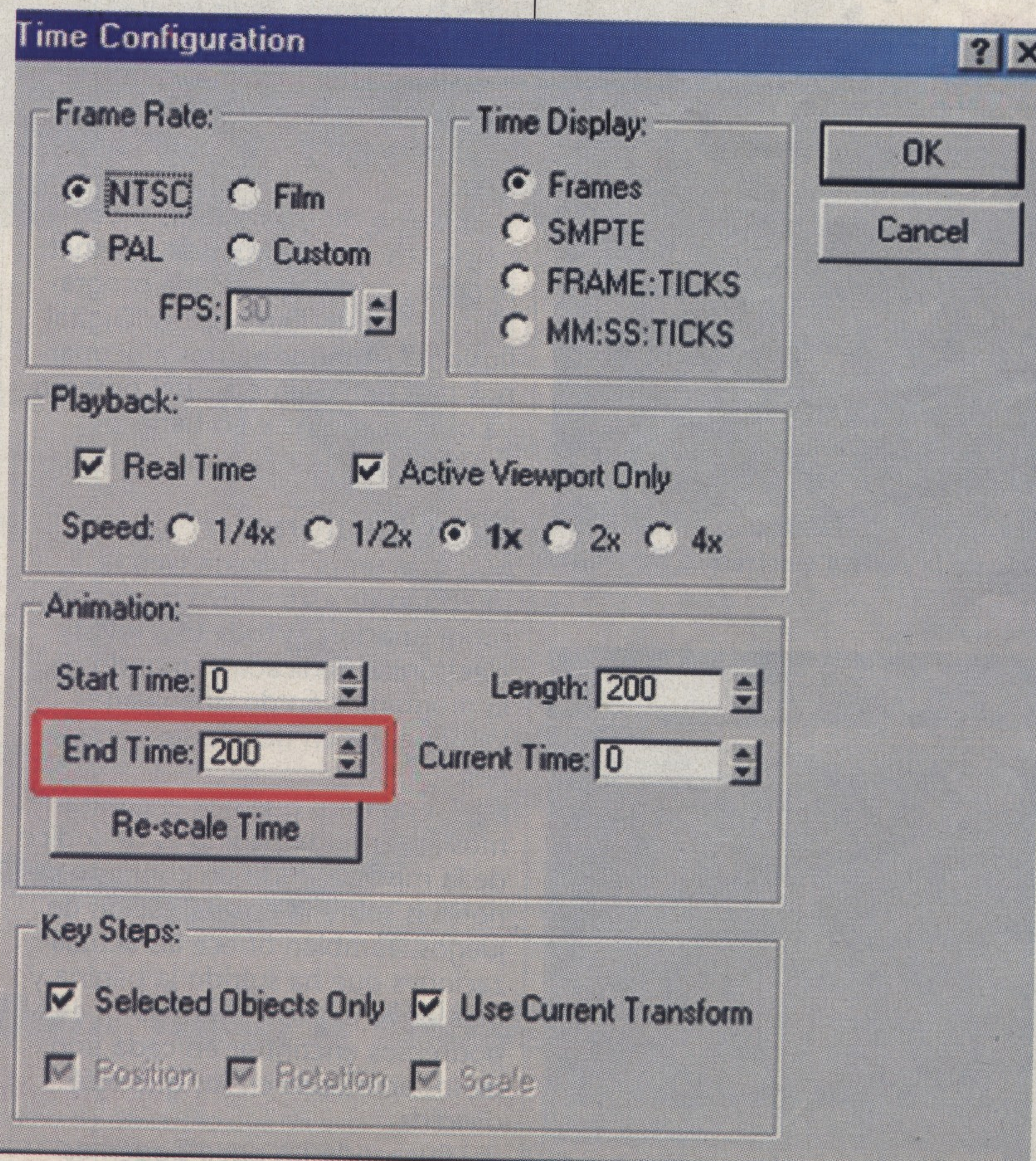


Figura 15 - Cambiando el nº de fotogramas

DIVagando por la Red

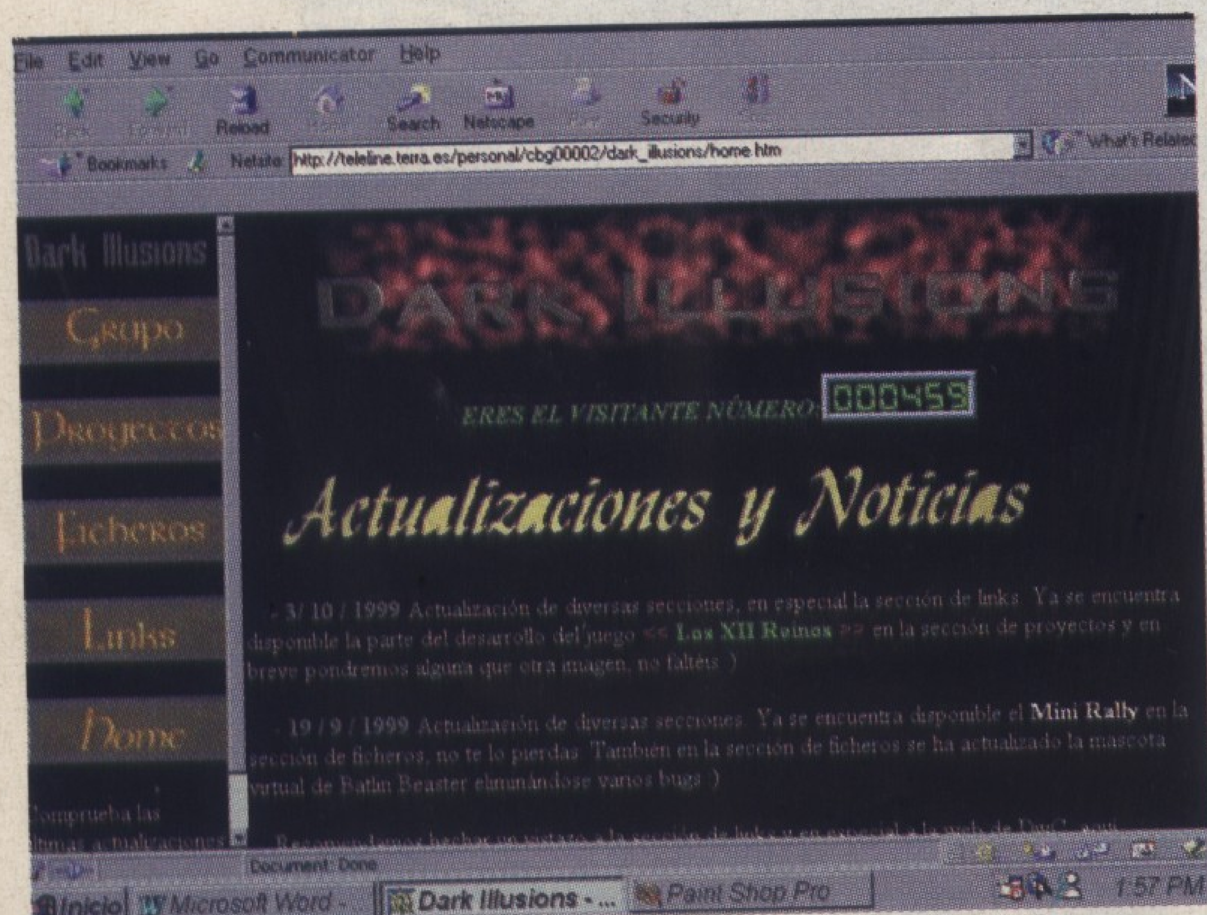
Nuevas páginas para nuevos proyectos

Están empezando a surgir multitud de grupos de programadores interesados en crear juegos para PC, de aficionados y de gente que ve en la programación de videojuegos una opción más que interesante para encauzar su futuro profesional. Aquí tenéis un par de muestras de estos jóvenes creadores.

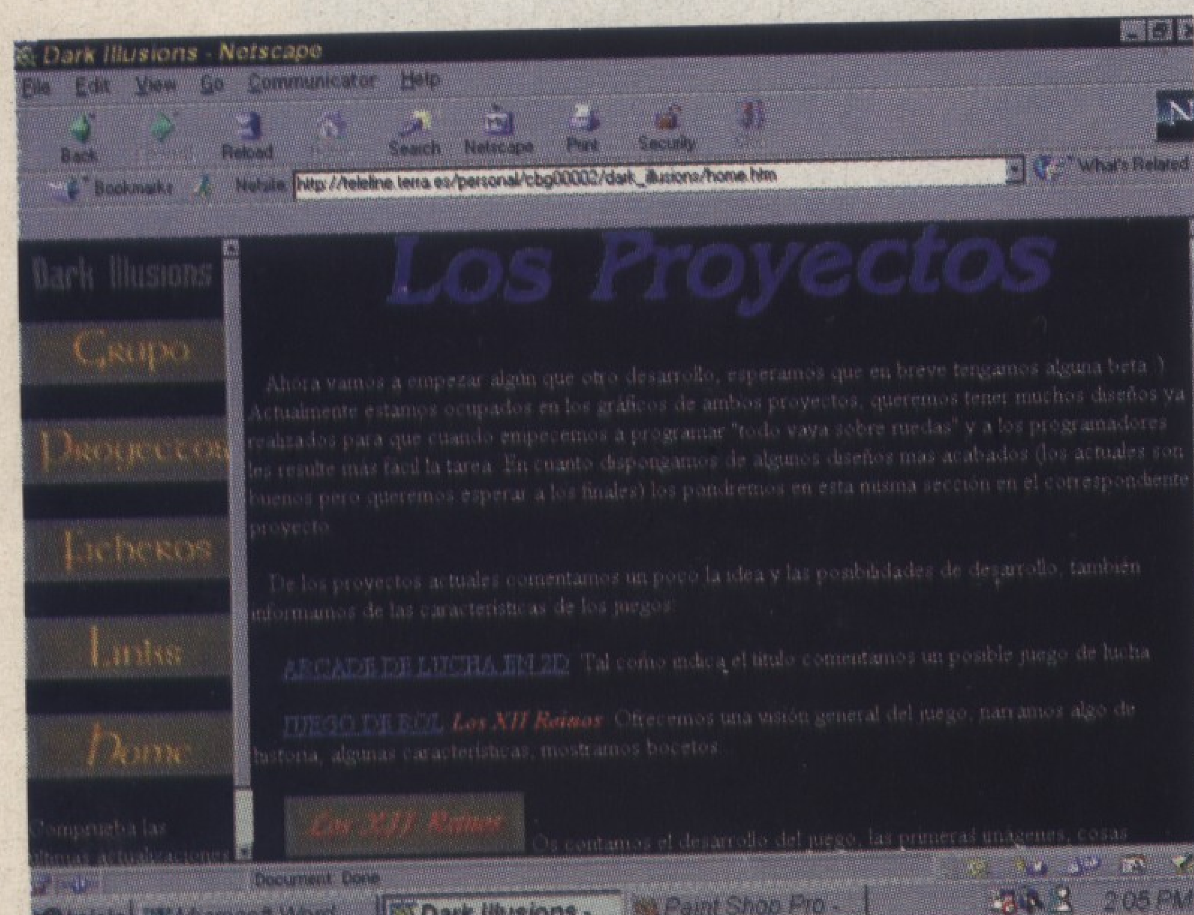
No nos cansamos de repetir que la programación de videojuegos es una profesión que dentro de nada será una de las ocupaciones más pujantes dentro del sector informático. Ya hay bastantes gurús que profetizan que la industria de los videojuegos hará sombra, en cuanto a volumen de negocio se refiere, al todopoderoso y luminoso sector del cine.

En España están surgiendo varios grupos de programación que se plantean ya sus primeros proyectos, unos a nivel de aficionados, otros pensando en vender su juego a la distribuidora de turno. Tanto unos como otros dedican gran parte de su tiempo al noble arte de trasladar su fértil imaginación a la pantalla del ordenador para que otras personas disfruten de su trabajo.

En este número vamos a ocuparnos de dos páginas de Internet hechas por dos grupos de programación: "Dark Illusions" y "Digital Ilusions". Aunque vamos a ocuparnos más extensamente del primero ya que su página web tiene "miga".



Página de inicio de la revista electrónica de "Dark Illusions".



Aquí podrás encontrar los próximos proyectos de "Dark Illusions".

Dark Illusions

Más que simple página web la dirección de este nuevo grupo de programación es toda una revista electrónica dedicada al mundo de la programación de videojuegos. Vamos a ver en detalle sus secciones.

Lo primero que nos encontramos al cargar la página es un índice de la misma y todo un conjunto de noticias sobre la programación de juegos. También ofrece las actualizaciones que ha sufrido la página y una breve explicación de lo que podremos encontrar en cada uno de los apartados en los que está dividida.

Buceando un poco podemos enterarnos de que este grupo de programación esta formado por varios componentes de 18 a 20 años y, ¡oh, sorpresa!, por un joven programador de ¡tan sólo 10 años! ¿Estaremos ante un John Romero o Sid Meier español? El tiempo lo dirá, desde luego es una edad muy temprana para programar, pero en esto pasa como con el aprender a nadar, mientras más temprano mejor.

A todos les gustaba programar y se conocían entre sí antes de plantearse formar un grupo para crear proyectos conjuntos. Y es que, si se quiere hacer algo de calidad, el programador solitario es un bicho raro en este negocio de la programación de software lúdico. El gran volumen de trabajo que exige cualquier proyecto medianamente aceptable hace que los grupos de trabajo, algunos bastante numerosos, sean de lo más habitual.

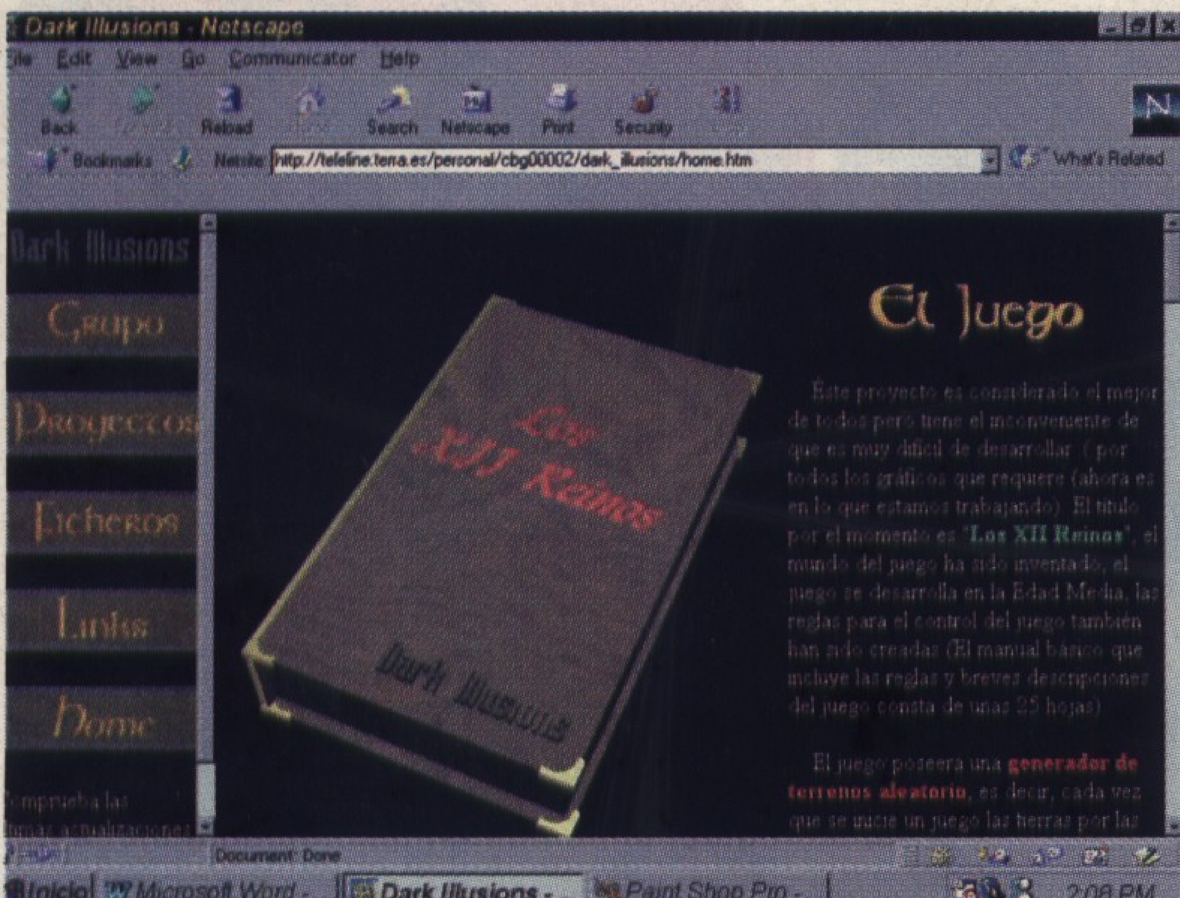
También nos cuentan los problemas a los que se enfrentaron al inicio de su andadura y nos ofrecen unas cuantas direcciones de correo para poder contactarlos. Sus apodos son los siguientes: "The Machine", "Batlin Beaster", "Lord Atlas", "Dark Wing" y "D1360". Un saludo a todos.

Proyectos, proyectos...

El grupo "Dark Illusions" tiene ahora mismo dos proyectos en cartera. El primero es un arcade de lucha en dos dimensiones, parecido a los clásicos juegos tipo *Street Fighters*. La idea es crear personajes que se salgan de lo corriente y no se parezcan a los habituales del subgénero del combate cuerpo a cuerpo. Piensan crear a los protagonistas de este videojuego con más de cuarenta movimientos de ataque y defensa, e introducir todos los efectos especiales que sean posibles para darle vistosidad al juego.

El segundo proyecto, que es el que tienen más avanzado, llevará por nombre "Los Doce Reinos". Es un juego de rol pero que contará con características propias no muy





Juego de rol "Los 12 Reinos".

usuales en este tipo de juegos y que, por supuesto, se mantienen en secreto.

Esta ambientado en la Edad Media y tienen incluso un manual de 25 páginas con las reglas que deberá seguir el que se adentre en esta tenebrosa época de la historia de la humanidad a través de "Los Doce Reinos". El juego contará con un generador aleatorio de mapas o escenarios, cada vez que se inicie el juego cambiará el terreno por donde se moverán los personajes. También contará con un generador aleatorio de los objetivos a cumplir en cada misión. Su argumento lo podréis encontrar en el cuadro adjunto a este artículo.

Los grupos de desarrollo españoles que empiezan explican sus futuros proyectos en Internet

Las dificultades con las que se encuentra este grupo para llevar su nave a buen puerto son los siguientes: primero, el gran volumen de trabajo gráfico que conlleva un juego de este tipo si quiere hacerse mínimamente variado; la segunda, los problemas para convertir el juego en un título que soporte un uso multijugador a través de Internet. Con *DIV 2* han logrado programar el juego para que soporte el juego simultáneo de hasta 16 personas en red local y la posibilidad de que lo disfruten dos usuarios conectados por un módem. El problema de compatibilidad con Internet lo están intentando solucionar con otros programas.

Están a punto de ofrecer las primeras imágenes de este juego en su página, quizá para cuando leáis estas líneas ya estén disponibles. También esperan poner a disposición de sus visitantes una versión beta de "Los Doce Reinos" en cuanto acaben el grueso del trabajo, para que los usuarios les den su opinión sobre el juego.

La página de "Dark Illusions" también posee una sección de descargas, con algún que otro juego completo en forma de código fuente con tutoriales y los ficheros necesarios para compilarlo. Podemos también ver el primer programa que realizó nuestro amigo de 10 años.

Este es argumento de "Los Doce Reinos":

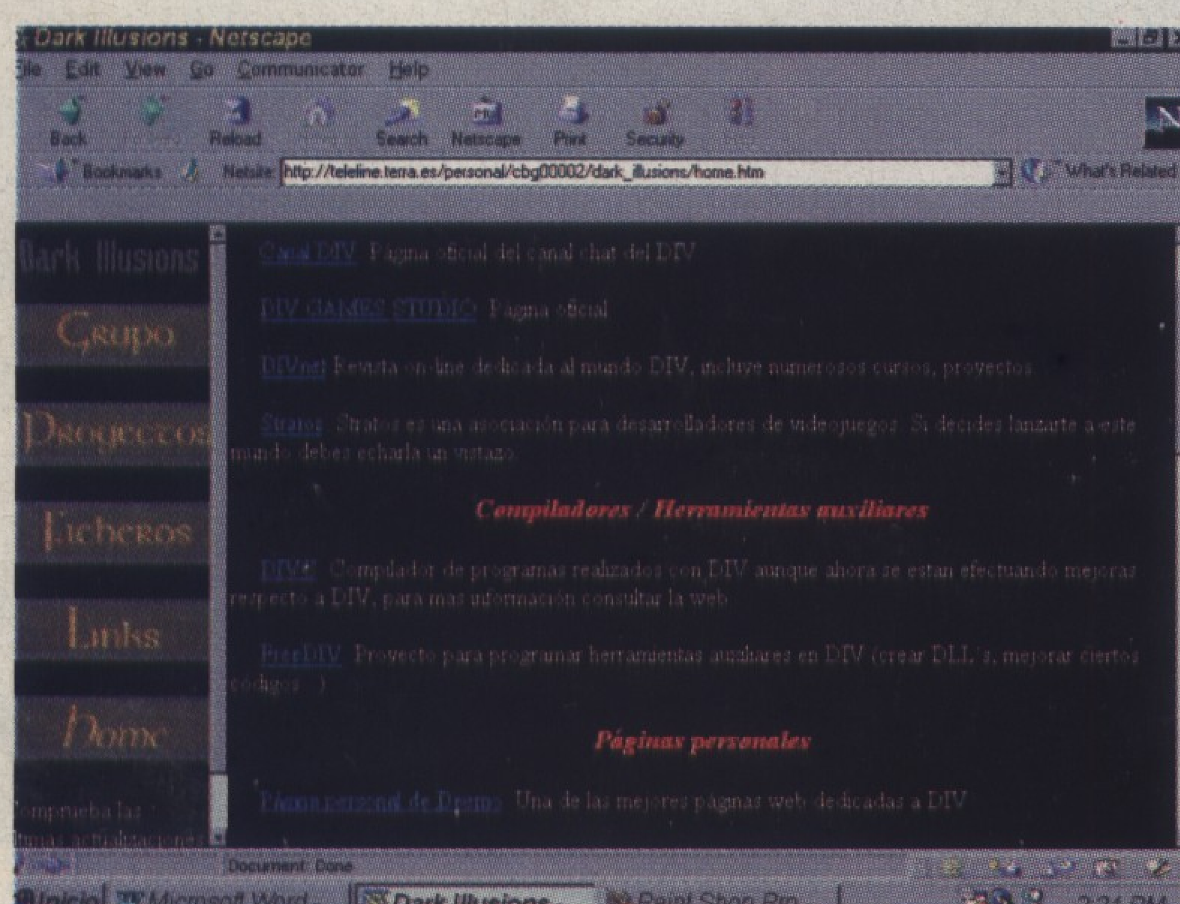
"Nos encontramos en la Edad Media, estamos en el Mundo de los Señores, un mundo en el que habitan 12 razas distintas: Duendes, Elfos, Enanos, Goblins, Hadas, Humanos, Junedaicos, Ogros, Orcos, Semielfos, Trishóculis y Trolls. Actualmente cada raza habita en un reino propio después de las grandes guerras por la independencia y la supervivencia.

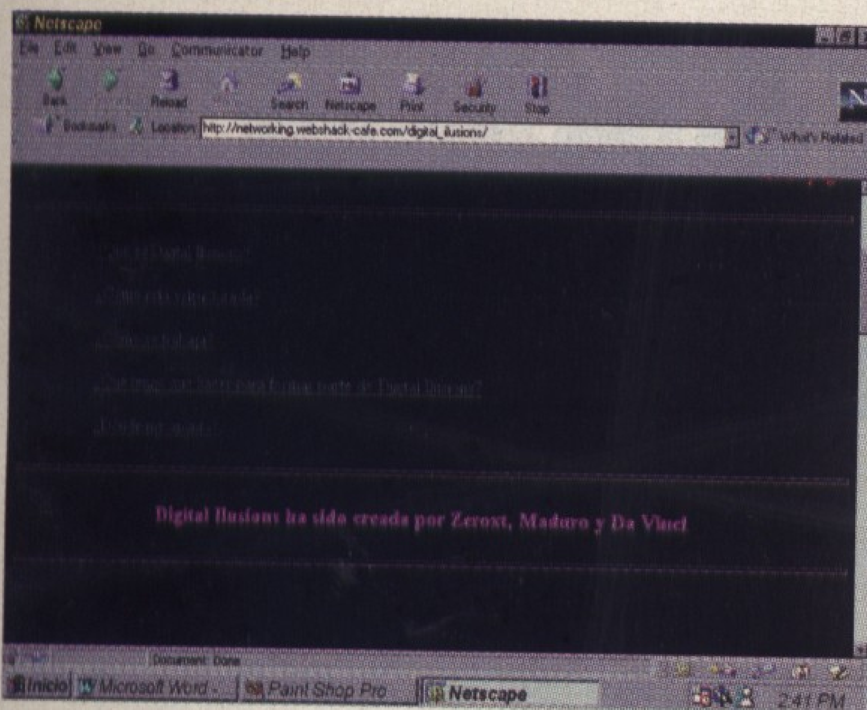
"Hace no mucho tiempo existía únicamente un reino, en el cual habitaban todas las razas, con sus pormenores y pormayores, hasta que el Señor Oscuro decidió ser el amo y señor de todas las razas; para llevar a cabo su tarea decidió liderar algunas razas de mentalidad débil. Logró dominar a los Duendes, Goblins, Ogros, Orcos y Trolls. Tras esto se sucedieron las terribles guerras por la dominación del reino, a medida que el Señor Oscuro intentaba con sus hordas conquistar a las demás razas éstas se arrinconaban y luchaban por su supervivencia.

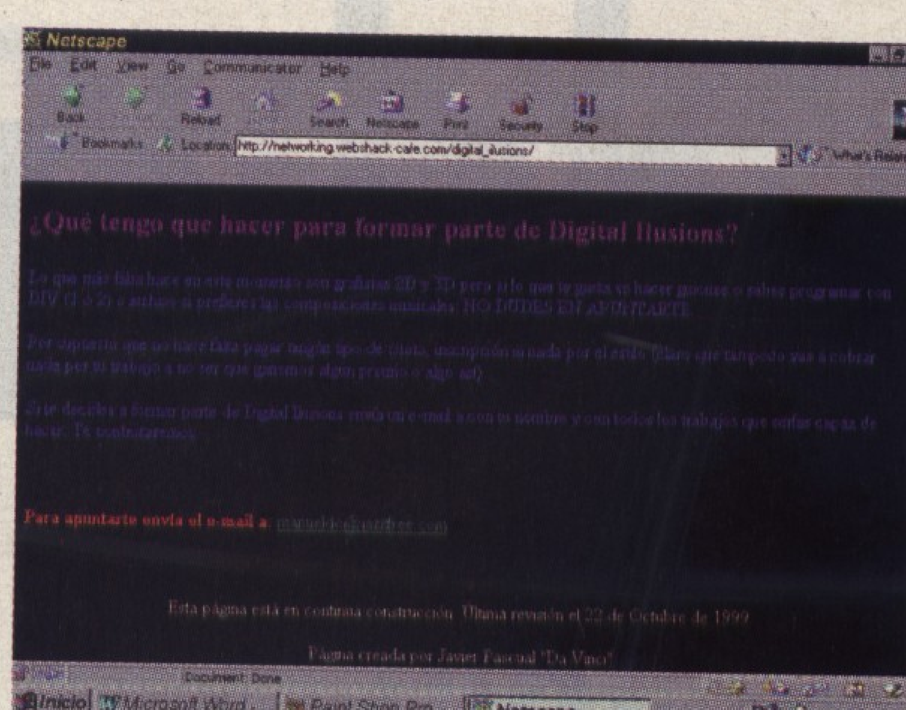
"Después de unos cuantos años de guerras mortales, la tierra se dividió en varios reinos: Reino de los Elfos, Reino de los Enanos, Reino de las Hadas, Reino de los Humanos, Reino de los Junedaicos, Reino de los Semielfos, Reino de los Trishóculis y Reino del Señor Oscuro (era el más grande y poderoso).

"Los ejércitos del Señor Oscuro decidieron someter a los Trishóculis y tras largas contiendas lograron dominarlos al final. Mientras el Señor Oscuro intentaba conquistar a los Trishóculis, las demás razas decidieron hacer una alianza cuyo único fin era poder defenderse del Señor Oscuro. Para ello, en caso de necesidad, cada reino movilizaría tropas contra el Señor Oscuro. De ésta forma se crearon lo que hoy en día se conocen como los Reinos de los Cielos. Las hordas del Señor Oscuro acabaron por sublevarse contra éste y lograr su semi-independencia (no dependían directamente del Señor Oscuro pero se encontraban bajo sus dominios), formándose otros 6 reinos conocidos como los Reinos de la Oscuridad".

Para descargarte todo lo que quieras.







Por último decir sobre esta página que tiene una más que interesante sección de enlaces o links.

Digital Ilusions

Y seguimos colmando nuestras "ilusiones", de "ilusiones oscuras" pasamos a "ilusiones digitales". "Digital Ilusions es el título que lleva otra página web, creada por un grupo de programación comandado por Zeroxt, Maduro y Da Vinci.

En esta dirección web nos explican que llevan poco tiempo en esto de la programación, pero que hay posibilidades de empezar a desarrollar un juego de estrategia en tiempo real como primer trabajo.

Parece que la labor de organización de los distintos subgrupos de desarrollo está bastante adelantada, como tiene que ser si se intenta dedicarse profesionalmente a este negocio. Intentan dividir el trabajo en cuatro secciones: una dedicada a realizar el guión del juego, un subgrupo dedicado a la parte gráfica, otro volcado de lleno en la programación, sin olvidar otro conjunto de personas para el apartado sonoro y musical. En total esperan contar con unas 25 personas. Desde luego con 25 personas trabajando para realizar un solo juego se pueden hacer grandes cosas.

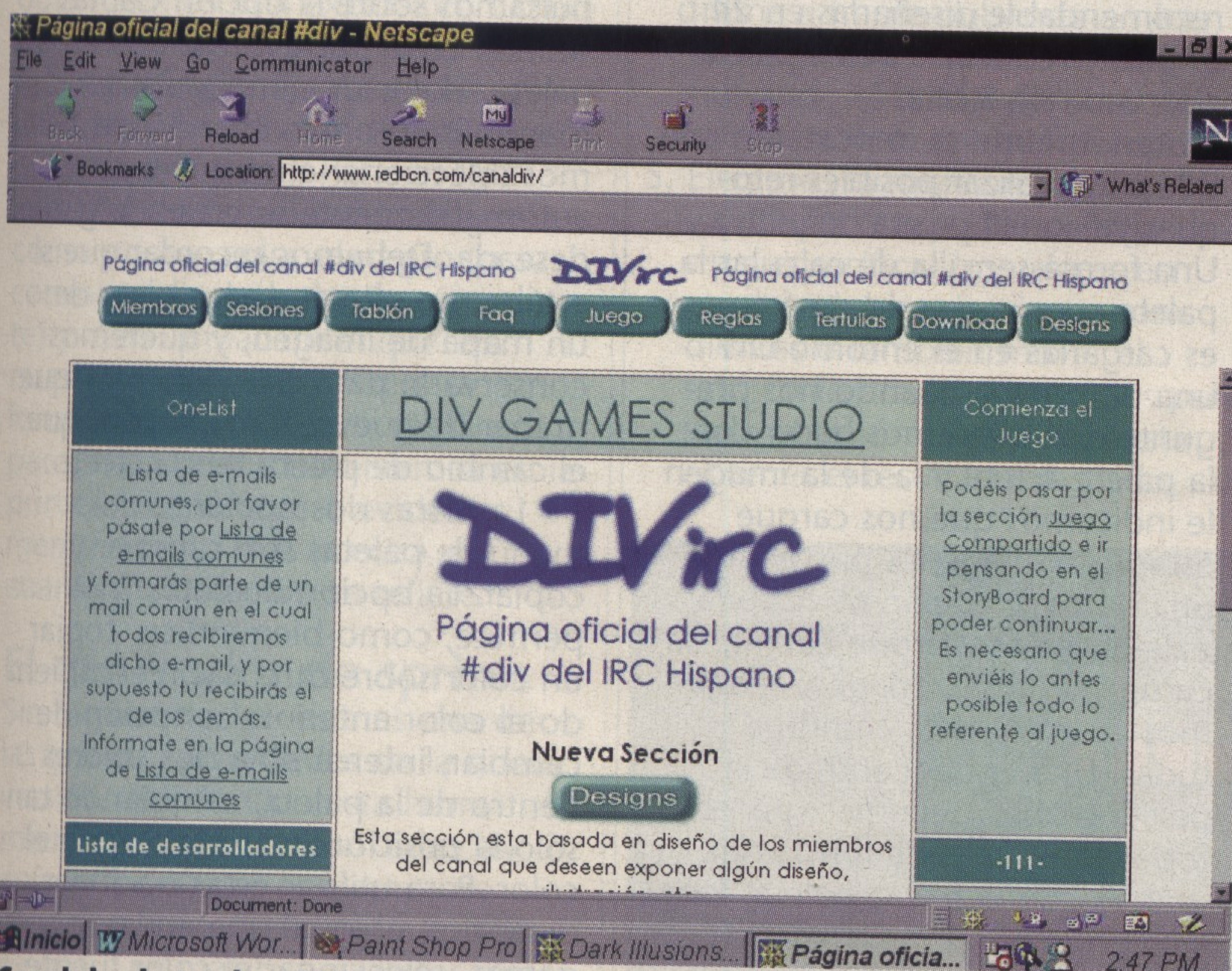
Este grupo tiene claro que las decisiones importantes, por ejemplo que proyecto se va a desarrollar, serán elegidas por mayoría absoluta de todas las personas que integren el grupo de desarrollo.

Tienen ya realizado el plan de trabajo que deberán seguir todos los subgrupos. Todo muy profesional, y lo más interesante: están buscando a gente que quiera unirse a su proyecto. Buscan sobre todo a grafistas 2D y 3D, programadores que sepan utilizar *DIV* o *DIV 2* y músicos para realizar composiciones musicales. Sólo tienes que enviarles un e-mail si estás interesado a esta dirección: manueldo@jazzfree.com.

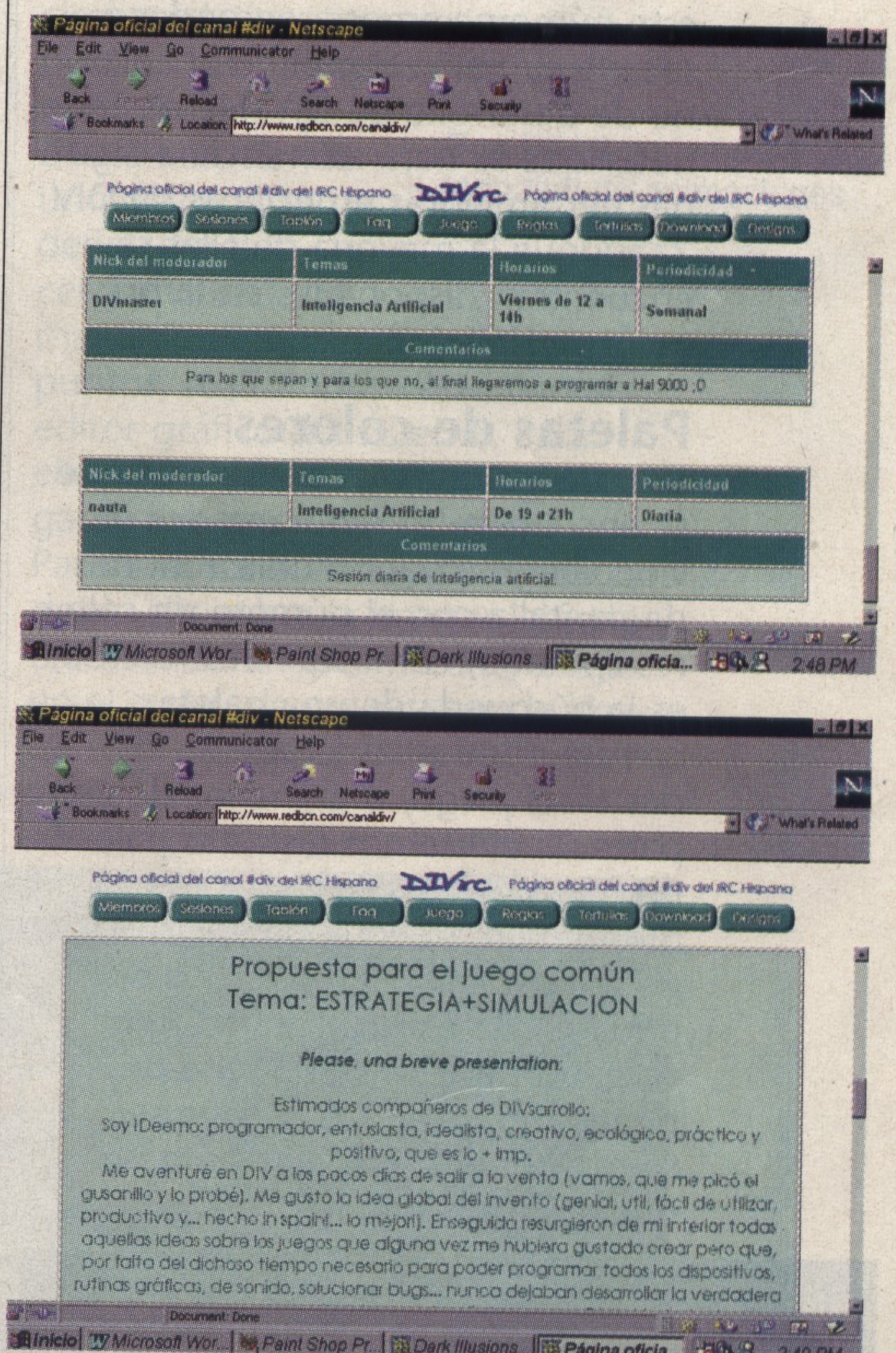
Nos ha quedado por comentar el primer canal de chat dedicado a la programación de juegos con *DIV* y *DIV 2*. Podéis ver alguna pantalla en este artículo, a modo de entremés, ya que nos ocuparemos de esta dirección de Internet en próximos números. Si no puedes esperar a que lo hagamos no dudes en entrar en la dirección que te damos en el cuadro adjunto.

Si tienes ideas o te apasiona programar forma tu propio grupo o únete a otro ya en marcha

Alfredo del Barrio



Canal de chat sobre DIV.



Enlaces a visitar:

Revista electrónica del grupo de programación "Dark Illusions":
http://www.teleline.es/personal/cbg00002/dark_illusions/home.htm

Grupo de programación "Digital Ilusion":
http://networking.webshack-café.com/digital_ilusions

Canal de chat sobre DIV:
http://www.redbcn.com/canaldiv

El manejo del entorno

Los aspectos gráficos en DIV

Desempolvemos nuestro espíritu aventurero y exploremos el entorno de DIV. Descubriremos multitud de entresijos que harán a nuestros juegos desmarcarse del resto.

Hasta ahora hemos aprendido a, mediante algunos recursos sencillos que nos ofrece DIV, realizar juegos con efectos de diversa índole. En este número vamos a aprender a sacarle el máximo partido al entorno de desarrollo de DIV. Hemos de notar que se hará referencia a algunas herramientas específicas de DIV 2, aunque la mayoría de ellas se encuentran disponibles en la primera versión.

Paletas de colores

Uno de los grandes problemas que se encuentra un programador de videojuegos cuando utiliza modos de pantalla con el número de colores limitado, como es nuestro caso, es la búsqueda de una paleta común a todas las imágenes usadas en el juego, o al menos de una parte. Si utilizamos distintas paletas, para objetos que aparecen al mismo tiempo en la pantalla, podemos obtener resultados sin duda no deseados. Para arreglar esto, proponemos una serie de puntos que bien tienen que ser tomados en cuenta a la hora de diseñar imágenes:



El editor gráfico en acción.

- Es conveniente agrupar las imágenes que van a aparecer al mismo tiempo en la pantalla, de forma que podamos encontrar paletas comunes a estos grupos y no a todas las imágenes a la vez, optimizando enormemente los resultados, y obteniendo imágenes más cercanas a las iniciales.
- Si a la hora de crear imágenes, utiliza herramientas externas, es recomendable diseñarlas en 24 ó 32 bits y reducirlas a una paleta adecuada de 8 bits. Posteriormente se importarán con DIV para realizar posibles retoques ya con la paleta definitiva.
- Una forma sencilla de calcular la paleta común a varias imágenes es cargarlas en el entorno DIV una por una, y cuando nos pregunte qué queremos hacer con la paleta actual y la de la imagen le indicamos que nos cargue ésta. Para imágenes posteriores, utilizamos la opción de fusionar paletas, de forma que cuando se cargue la última imagen, tengamos ya una paleta común a todas. Una vez encontrada la paleta global, debemos grabar todos los mapas de forma que se graben con la nueva paleta. Para usuarios de la primera versión de DIV, otro método válido sería grabar una por una las paletas de los distintos mapas y fusionarlas, para después cargar la obtenida. Abriremos una por una las imágenes y las adaptaremos a la paleta actual.
- Tal vez la forma más profesional y que mejor resultados obtiene es la de utilizar herramientas externas específicas, como puede ser *Palette Works*. Con esta herramienta se nos permite especificar una lista de archivos de imagen a los que queremos encontrar una paleta común, que posteriormente nos devolverá en un archivo o nos convertirá cada imagen a dicha paleta.

Una gama de color es una serie de colores graduados que guardan un orden lógico de crecimiento o decrecimiento en sus valores RGB. En numerosas ocasiones desearemos realizar estas gamas o gradientes de color, sobre todo para crear diversos efectos visuales como pueden ser efectos de fuego y plasmas entre otros. Para ello debemos en primer lugar editar los colores extremos de la gama con el editor de paletas de DIV. Una vez realizado, se selecciona uno de ellos y pulsamos sobre la opción Gamas situado en la parte inferior de la paleta. El cursor cambiará a una mano. Es entonces cuando debemos seleccionar el otro extremo, y automáticamente se creará la gama deseada. Debemos recordar que si estábamos alterando la paleta de un mapa de imagen, y queremos conservarlo para éste, tenemos que grabar de nuevo el mapa para que el cambio de paleta tenga efecto.

Las otras dos opciones del menú de paletas son cambiar y copiar. La opción de copiar nos permite, como bien indica, copiar un color sobre otro, sobreescribiendo el color anterior. La opción de cambiar, intercambia dos colores dentro de la paleta, cambiando tan sólo la posición (su índice) y no el color. Para utilizar estas opciones, debemos, al igual que en las gamas, seleccionar el color fuente, pulsar sobre la opción correspondiente y seleccionar el color de destino.

Generador de colisiones

Una vez conocido el concepto de gamas de color, podemos entrar en profundidad en el nuevo generador de colisiones. Ya vimos en nuestro segundo número cómo crear colisiones con la primera versión de DIV. En la segunda, se han introducido algunos cambios, que



La barra de efectos especiales sobre una selección.

utilizan las gamas de colores. Como dijimos en su momento, podemos indicar el tamaño de cada uno de los mapas de la animación, así como el número de fotogramas, parámetros que permanecen invariables. Sin embargo desaparecen esos tres colores con los que definíamos la colisión. Ahora se utilizan las gamas de color. Si pulsamos sobre la barra correspondiente a dicha gama, aparecerá una ventana donde podemos alterar la gama. El método es distinto al usado en el menú de paletas. Pulsamos sobre el color que queramos cambiar, y una vez definidos, pulsamos sobre los recuadros inferiores de los que deseemos sean los extremos de nuestra gama. De esta forma podremos ver en la parte superior el resultado obtenido.

Si utilizamos por ejemplo una gama de rojo a amarillo (clásica explosión) e indicamos uno de los tipos de explosión (A, B o C, de más homogéneo a más disperso) obtendremos una explosión. Como podremos observar, se generará una colisión que finalizará en un cuadrado de color rojo y no tiene color transparente. Para arreglar esto, debemos indicar en la gama como primer color el negro (o su correspondiente color transparente) y a continuación el que era nuestro primer color. De esta forma obtenemos una colisión parecida a la que obteníamos en la primera versión, pero evidentemente mucho más precisa en cuanto a coloración se refiere.

El generador de sprites

Sin lugar a dudas, esta es una de las grandes novedades de la nueva versión de DIV. Con esta herramienta podemos crear todas las animaciones que queramos de modelos humanos para nuestros juegos. Los más beneficiados son los juegos de plataformas, que encuentran en esta herramienta una forma rápida, sencilla y efectiva de crear sus personajes. Para crear nuestros personajes, debemos seguir unos pasos determinados:

1. En primer lugar debemos crear lo que será el mapa base de texturas de nuestro personaje. Para poder guiarnos en cómo crearlos, es aconsejable ver los modelos que se incluyen de ejemplo. Como podremos apreciar, podemos crear las texturas para el cuerpo, la cara, la cabeza y el arma en caso de poseerla. Podemos utilizar para editarlo también cualquier programa de diseño que acepte los formatos JPG, PCX o BMP. La creación de estas texturas dependerán de nuestro propósito y de nuestro arte en el diseño de imágenes.
2. Una vez diseñado el modelo deseado (o escogido uno de entre los ejemplos) debemos elegir entre tres modelos: hombre, mujer o niño, así como si estará o no armado.
3. Ahora debemos determinar el tamaño del personaje en porcentaje. Debemos experimentar con unos cuantos valores hasta encontrar el ideal.
4. Elegimos de entre la lista la animación que deseamos. Es importante saber que la lista es distinta para los personajes armados y los que no. Para ver la animación, podemos hacerlo frame a frame pulsando en los botones con flechas dobles o pulsando el botón derecho del ratón sobre la imagen del sprite.
5. Elegimos la rotación y perspectiva del personaje. Para ello podemos especificar los valores en grados en las cajas correspondientes o también mover el ratón mientras mantenemos pulsado el botón izquierdo, hasta obtener el resultado deseado.
6. Por último le indicamos de cuántas imágenes queremos que conste la animación. Ahora pulsamos el botón de aceptar y tenemos como resultado nuestra animación.

Con esto podemos generar de forma fácil la animación para cualquier tipo de personaje en los juegos de plataformas y otros muchos.

Puntos de control

Sin duda un factor importante, y que le habrá dado a más de uno auténticos quebraderos de cabeza, es la posición del punto central de un mapa. Cuando le indicamos a un proceso que su posición es la (0,0), no correspondía a la esquina superior izquierda como muchos supondrían, sino al centro de la imagen. Para evitar esto, y poder definir sobre qué punto se refieren dichas coordenadas, debemos definir lo que se denomina un punto de control.

Los puntos de control son posiciones relativas a un mapa que pueden ser utilizadas para cualquier fin que deseemos, siendo uno de ellos, la definición del centro virtual del mapa. Podemos definir hasta mil puntos, realizándose esto con el editor gráfico del sistema. Para ello, debemos en primer lugar cargar la imagen en el sistema. Pasaremos a editarla realizando un doble clic sobre la imagen abierta o pulsando la opción editar mapa en el menú de mapas, con la imagen seleccionada de entre todas las ventanas.

Con DIV 2 podemos crear todo tipo de animaciones para personajes

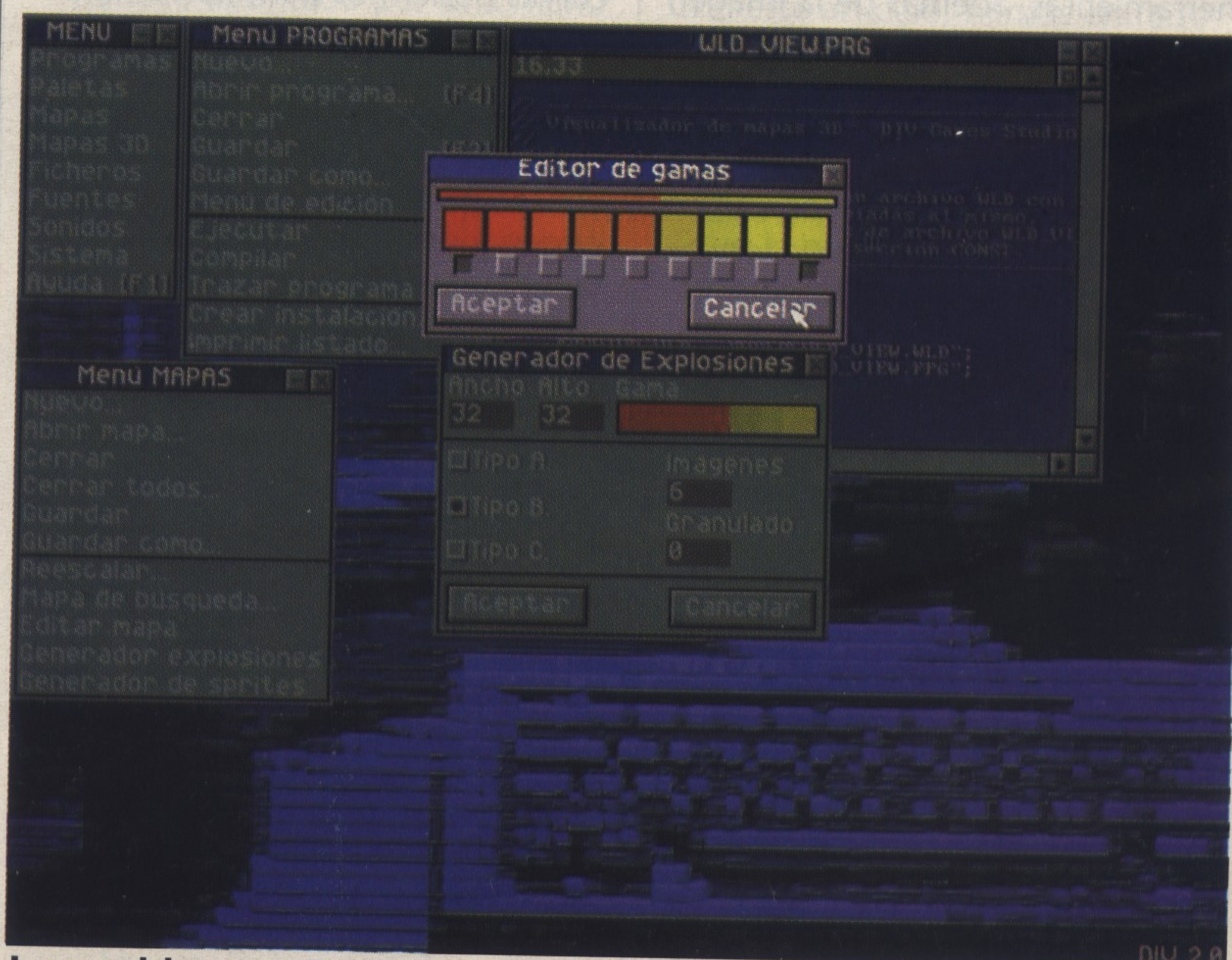


Imagen del nuevo editor de explosiones de DIV 2.

Una vez dentro del editor gráfico, debemos pulsar sobre la opción que está representada por 3 puntos con números a sus lados. En la barra inferior aparecerán diversos datos que son por orden de izquierda a derecha:

- Posición actual del cursor sobre el mapa.
- Tamaño de aumento de la imagen.
- Botón para pasar al anterior punto de control.
- Punto de control actual.
- Botón para pasar al siguiente punto de control.
- Posición del punto de control actual.

Elegiremos el punto de control número 0. Podemos observar que sus coordenadas son las centrales. Esto es porque el punto de control número 0 nos indica el centro virtual que queremos alterar. Coloquémoslo en el lugar que deseemos. Ahora podemos cerrar el editor gráfico pulsando el botón derecho.

Para terminar debemos grabar la imagen para que surta efecto. Esto tiene grandes aplicaciones, siendo fundamental para indicar el *hot-spot* de los cursores que usaremos en nuestro programa.

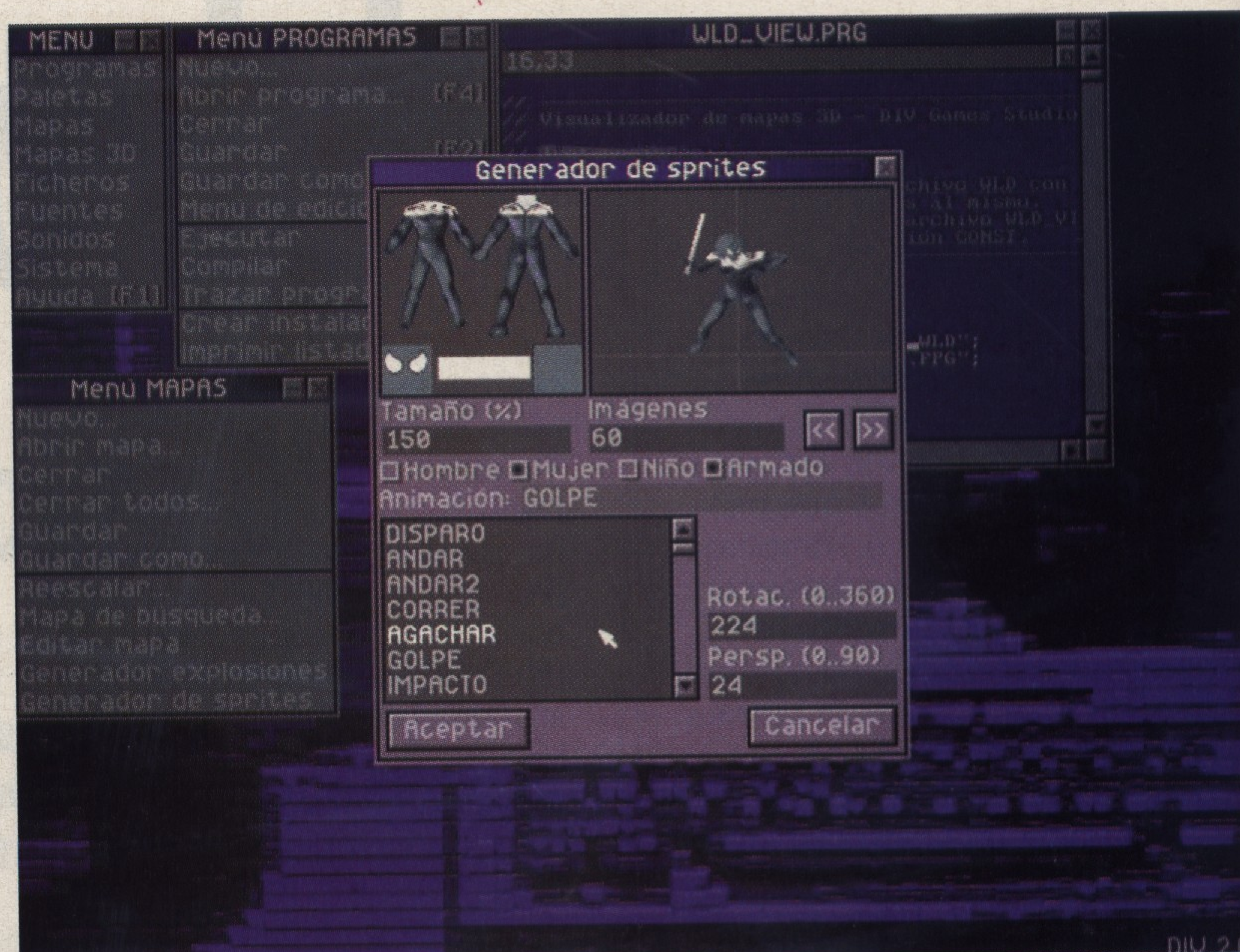
Con el editor gráfico podremos hacer uso de numerosas opciones

El editor gráfico

Hagamos un breve recorrido por esta útil herramienta, que posee potentísimas opciones. Para utilizarla sobre un mapa, debemos cargar éste y hacer doble clic sobre él o seleccionar la opción editar mapa del menú de mapas. Se abrirá un entorno que posee dos barras de herramientas, además de la imagen abierta a tamaño real. Veamos alguna de las principales opciones de que disponemos:

1. Dibujo de puntos: Para ello disponemos de dos opciones: el punteado y el lápiz. Con el punteado podemos situar en el mapa puntos individuales del color actual. Podemos controlarlo mediante el ratón o mediante los cursores y la barra espaciadora. Con él podemos retocar o realizar pequeños detalles en nuestras imágenes. Si deseamos crear trazos continuos usaremos el lápiz, herramienta mucho más cómoda para propósitos más generales. Si mantenemos pulsado el botón izquierdo del ratón, obtendremos a medida que desplazamos el ratón un trazo contínuo.

2. Líneas: Para dibujar líneas rectas, disponemos de otras dos opciones más. La primera de ellas, nos permite trazar una sola recta.



El generador de sprites, nuevo en esta versión de DIV.

Para ello pulsaremos el botón izquierdo del ratón donde deseemos que comience a dibujarse la línea. Desplazaremos el ratón hasta la posición que deseemos para el otro extremo de la línea y pulsaremos de nuevo el botón izquierdo para fijar la posición. La otra opción nos permite dibujar múltiples líneas rectas interconectadas formando una línea quebrada. Básicamente funciona de igual forma, sólo que cuando fijamos una recta, podemos fijar la posición del siguiente extremo de una nueva línea cuyo origen se encuentra en el extremo de la anterior. Cuando deseemos finalizar la línea quebrada, basta con pulsar el botón derecho del ratón.

3. Curvas: disponemos de dos tipos de curvas: bezier y splines. Las curvas bezier nos permiten definir un tramo curvo a partir de 4 puntos que debemos fijar: extremos, y dos pendientes. Las curvas splines, nos permite crear tramos curvos consecutivos fijando los puntos de origen y destino, junto con la pendiente de salida de la curva en el extremo, que se fijará pulsando las teclas + y -. No vamos a entrar en los detalles matemáticos de estas opciones, con lo que recomendamos que se practique con ellos para poder obtener resultados ajustados a nuestros deseos.

4. Rectángulos y elipses: con estas dos opciones podremos crear rectángulos mediante la definición de las esquinas superior izquierda e inferior derecha. Cuando pulsamos sobre esta opción aparecerá en la barra inferior una nueva opción donde podremos seleccionar entre un rectángulo relleno y otro perfilado. La opción para dibujar elipses, nos permite dibujarlas mediante dos tipos distintos de definiciones que podremos elegir en la misma opción de la barra inferior que en los rectángulos. La primera de ellas, nos permite dibujar una elipse definiendo los extremos del rectángulo que la inscribe. La segunda, crea la elipse a partir del centro de la misma y la definición de un extremo del rectángulo que la inscribe. Si queremos crear circunferencias, basta inscribir la elipse en un cuadrado.

5. Spray: con esta opción podemos dibujar como si de un aerosol se tratase, dibujando algunos puntos aleatorios con el grosor que le indiquemos en la opción que aparece en la barra inferior.

6. Relleno: permite rellenar zonas de un solo color de forma uniforme. Existen hasta cuatro tipos distintos de relleno, que no vamos a comentar por la extensión de éstos. Experimenta con ellos y verá sus capacidades.

7. Bloques: nos permite definir una región para aplicar sobre ella posteriormente un efecto. Tenemos varios tipos que podemos elegir en la barra inferior. Entre ellos destacamos las áreas rectangulares y las poligonales, que nos permiten elegir regiones mucho más precisas. La selección de éstas áreas se hace eligiendo los extremos del rectángulo para la primera opción (igual que en el dibujo de rectángulos) y para la segunda, eligiendo uno por uno los vértices del polígono hasta que pulsemos sobre el primero o pulsemos el botón derecho del ratón, momento en el que se unirá automáticamente el último y el primer vértice. Además de estos dos



tipos de selección básicos, disponemos de otros tipos de selección automática que no vamos a definir.

8. Deshacer: cuando pulsamos sobre esta opción, cambia el aspecto de la barra inferior, apareciendo cuatro nuevas opciones que nos permiten deshacer o rehacer las últimas acciones realizadas sobre el dibujo. Si pulsamos sobre las flechas simples desharemos o reharemos la última acción, si pulsamos sobre las flechas dobles, trataremos las acciones más rápidamente.

9. Texto: sirve para colocar texto en las imágenes. Si pulsamos sobre el mapa en un punto con esta opción seleccionada, se abrirá una ventana de selección donde irán apareciendo los caracteres a medida que los introduzcamos. Para borrar los caracteres, se hará como en cualquier procesador, con la tecla de *backspace*. Por defecto, se utilizará la fuente seleccionada para el editor de código de DIV. Para cambiar de fuente, debemos crear una fuente o cargar una ya definida mediante el menú de fuentes y tener activada su ventana.

9. Puntos de control: fueron ya comentados anteriormente.

10. Cuentagotas: esta imagen aparece cuando seleccionamos algunas de las herramientas anteriores. Si pulsamos sobre ella, aparecerá un nuevo cursor con el que podemos ver y seleccionar el color del punto sobre el que pulsemos.

Además de estas herramientas, disponemos en la zona inferior de la pantalla otra barra con diversas opciones, de entre las que destacamos:

- **Selección de opciones:** nos permite escoger una de las herramientas de las anteriormente citadas.
- **Coordenadas actuales:** indica las coordenadas actuales del cursor.
- **Lupa:** nos permite aumentar el tamaño hasta en 8 veces. Pulsando sobre él aparecerán los cuatro aumentos disponibles.
- **Deshacer:** nos permite deshacer la última acción sin pulsar sobre la opción de deshacer de la barra de herramientas.



Menú de edición de paletas.

- **Color actual:** pulsando sobre este recuadro, aparecerá la paleta de colores, de la que podremos seleccionar el color que deseemos para dibujar.
- **Texturas:** pulsando sobre la opción situada en la parte inferior del color actual (simbolizado con una especie de U), aparecerá una ventana con todos los mapas abiertos con los que podemos dibujar.
- **Color transparente:** simbolizado por una T, podemos seleccionar el color transparente simplemente pulsando sobre ella.
- **Gama actual:** gama de colores actual, o colores que podemos seleccionar de forma rápida sólo con pulsar sobre ellos.
- **Porcentaje:** aparece para algunas opciones de dibujo. Nos permite indicar mediante un valor de porcentaje, la cantidad de tinta que queremos añadir a la hora de dibujar en la pantalla. De esta forma podemos crear formas translúcidas en lugar de opacas.

Además de estas opciones podemos utilizar una serie de efectos especiales sobre algunas de las selecciones que hagamos con la herramienta de selección. Una vez seleccionada una zona o región sobre la pantalla, aparecerán una serie de nuevas opciones que a continuación describiremos:

- **Crear un nuevo mapa:** pulsando sobre el icono de una ventana, podemos crear un nuevo mapa en el escritorio con el contenido y las dimensiones de la selección.
- **Mover bloques:** seleccionando sobre la opción indicada por un muñeco apuntado por una mano aparecerán una serie de siete nuevas opciones:
  - *Opacidad:* permite seleccionar entre dos modos de copia: opaco y translúcido.
  - *Transparente:* cuando queremos que el color transparente (de índice 0) no se trate como tal, y se copie el bloque entero como un bloque rectangular.
  - *Inversión horizontal:* aplica un efecto de espejo sobre la selección.
  - *Inversión vertical:* invierte la selección verticalmente.
  - *Rotación:* una vez situada la imagen seleccionada sobre un nuevo punto, podemos aplicarle un giro que definiremos moviendo el ratón y pulsando de nuevo el botón izquierdo del ratón.
  - *Escalado:* amplía o reduce el bloque seleccionado.
  - *Borrado:* rellena la zona elegida con el color seleccionado actualmente en la barra.
- **Efectos:** simbolizado por las letras FX, en este menú podremos cambiar el contraste y la iluminación entre otros. Veamos sus opciones:
  - *Pasar colores a gama seleccionada:* convierte los colores del bloque seleccionado a los de la gama seleccionada en la barra de herramientas.
  - *Invertir:* invierte, es decir, crea el negativo de la región seleccionada.
  - *Crear borde:* crea un borde de un punto de grosor con el color que esté actualmente seleccionado.
  - *Aclarar:* incrementa la luminosidad de la imagen.
  - *Oscurecer:* decrementa la luminosidad de la imagen.
  - *Suavizar:* aplica un efecto de blur o suavizado de la imagen.

Crear efectos especiales es una de las posibilidades que ofrece DIV

Además de estos efectos es interesante conocer la existencia de un efecto parecido a este último de suavizado mediante las herramientas de dibujo. Consiste en mantener pulsada la tecla D mientras que se dibuja con el lápiz, el spray o las líneas, de forma que en lugar de dibujar con el color actualmente seleccionado, aplicará un efecto de suavizado sobre la imagen. De esta forma, podemos por ejemplo suavizar el escalado producido en las líneas de mucha pendiente, o los contornos de un dibujo hecho a mano.

Conclusión

Con este artículo, hemos entrado en otro aspecto importante en el desarrollo de los juegos, y es el aspecto gráfico. DIV, no ajeno a esto, posee un entorno completamente integrado con el que podemos hacer auténticas maravillas. El único límite está en el dibujante y la imaginación. No obstante, es recomendable el utilizar unos gráficos de base, sencillos y no muy elaborados, para así poder abordar el código y lo que es el juego en sí mismo, para una vez terminado, poder ya desarrollar unos dibujos definitivos y de mayor calidad.

Me despido por este número, animándoles de nuevo a experimentar por sí mismos y a descubrir nuevas tierras en este amplio mundo de DIV. Para cualquier consulta pueden dirigirse a la dirección de correo electrónico trinidad@arrakis.es.

Pablo Trinidad

Salvapantallas y DLLs de autocarga

Qué son y cómo funcionan

En los números anteriores nos hemos centrado en la creación de funciones para ampliar la funcionalidad de DIV. Alejándonos un poco del concepto de utilidad, estudiaremos cómo crear salvapantallas que alegren nuestros juegos, además de las siempre interesantes DLLs de autocarga.

Los salvapantallas nacieron como una necesidad con aquellos casi extintos monitores de fósforo. Si una imagen aparecía estática durante un largo período de tiempo, podía degradarse el monitor y mantener de forma perenne la imagen. Por ello, se utilizaban los denominados salvapantallas, que ponían en negro la pantalla. En los monitores actuales, al funcionar con haces de electrones, no tienen este problema de perpetuidad de colores, con lo que el uso de los salvapantallas se convierte en algo caprichoso y meramente estético.

Como en los juegos es bastante importante la estética, podemos querer en algún momento disponer de un salvapantallas. Por supuesto, DIV no sólo nos lo permite, sino que además nos lo facilita con funciones específicas para su construcción. Crearlos por tanto ya está al alcance de cualquiera. Veamos todos los fundamentos que envuelven a la creación y funcionamiento de estos pequeños programas informáticos.

¿Cómo funciona un salvapantallas?

Tomemos como ejemplo cercano el funcionamiento de un salvapantallas en Windows. En realidad no son más que meros programas ejecutables como otro cualquiera que reciben un parámetro en la línea de comandos que le indica cómo debe actuar cuando transcurre un determinado tiempo de inactividad del sistema, llamado tiempo de espera. Estos programas muestran durante su corta vida un determinado efecto gráfico, frecuentemente acompañado de música o efectos de sonido. Cuando detecta la pulsación de una tecla, el movimiento del ratón o del joystick, el salvapantallas se finaliza, dando paso de nuevo a la aplicación que hasta antes de ejecutarse estuvo en uso. Pues bien, en DIV no nos quedamos muy lejos de este funcionamiento. Tal vez la única diferencia se encuentre en el formato y carga de éstos, que recordemos serán DLLs o librerías de enlace dinámico de DIV de autocarga. Estas librerías se activaban automáticamente si al ejecutarse el programa éstas se encuentran en el mismo directorio que el ejecutable. Como veremos, esto debemos indicarlo de alguna manera en el código del salvapantallas.

Si deseamos probar el funcionamiento de un salvapantallas desde el propio entorno de DIV tan sólo debemos copiar el archivo DLL al directorio donde tengamos instalado el DIV.

Creación de salvapantallas

Para crear un salvapantallas, necesitamos implementar una serie de funciones y variables que a continuación detallamos:

- *ss_init()*: es la función de inicialización del salvapantallas. DIV la llamará cada vez que salte el salvapantallas, y nos servirá para fijar el tiempo de espera del salvapantallas, así como para reservar memoria, crear valores predeterminados y en definitiva todas aquellas operaciones que necesitemos para que nuestra rutina de dibujado funcione correctamente.
- *ss_frame()*: esta función se ejecuta cada frame, y es la que nos permitirá realizar una animación o cualquier efecto dependiente del tiempo sobre la pantalla. Es el equivalente a post_process_buffer en las DLLs de funciones, y nos permite realizar operaciones con el buffer de vídeo antes de cada volcado sobre la pantalla.
- *Ss_end()*: en ella realizaremos todas aquellas operaciones como liberar memoria o cerrar archivos



DIV developer

NÚMERO 5

Sumario

Curso de programación y concurso

En este número cinco de Divmanía hemos pasado por bastantes dificultades para elegir qué tres obras se llevarían los premios del concurso de programación de juegos. Parece que el nivel está subiendo de día en día. Y no sólo la calidad mejora, la cantidad de juegos que recibimos es cada vez mayor también. Algunos se nota que son los primeros trabajos de programación, pero todos suelen tener algún punto fuerte por el que merecerían ser publicados, y puede que lo hagamos. Estamos pensando en hacer un número especial comentando todos estos juegos y metiéndolos en el CD-Rom que acompaña a la revista. Así que animaros, todos los juegos son bien recibidos.

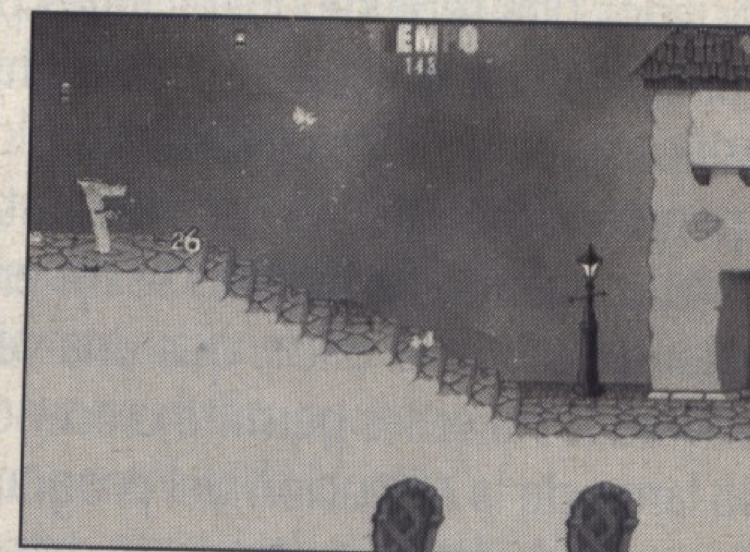
En cuanto a los cursos de este DIV Developer, continuamos con los siguientes capítulos iniciados en el número anterior de nuestra revista. Como ya sabéis, nuestro primer curso trata de introducir a la programación a los no iniciados. Para ello continúa su repaso a la programación básica, es decir la programación basada en algoritmos.

El segundo curso también sigue la línea del número anterior introduciéndonos a la programación en C, un lenguaje de programación que ya es algo más que un clásico, aunque mantiene su vigor gracias a sus excepcionales características. Es muy importante conocer este lenguaje porque es uno de los más usados para la realización de todo tipo de juegos.

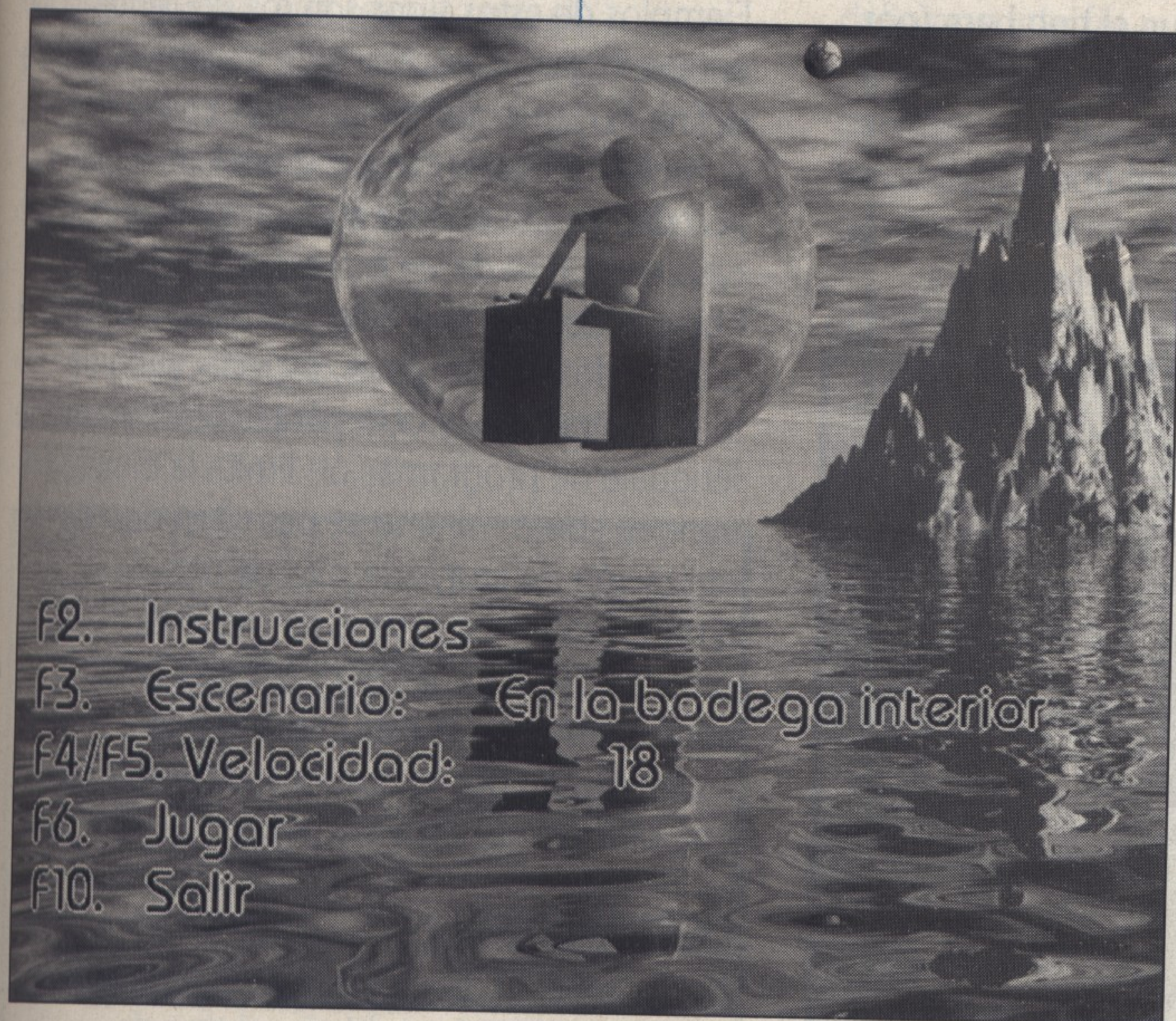
Por último, tenemos el curso de programación en ensamblador donde se continúa dando cuenta de todo lo que tiene que ver con el lenguaje que utilizan los microprocesadores de nuestros ordenadores. Si quieres saber cómo piensan las máquinas esta serie de artículos te ayudarán a conseguirlo.

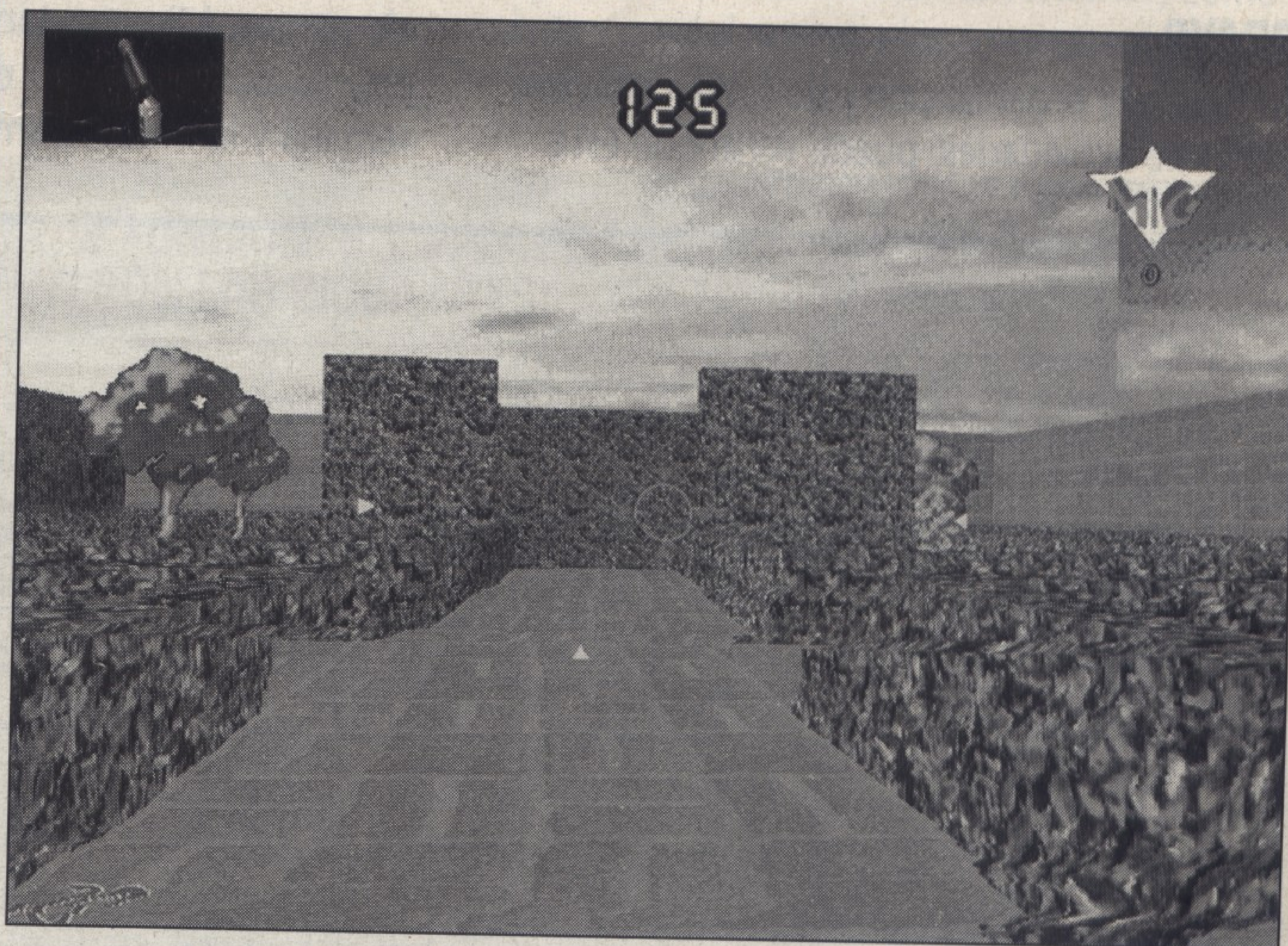
Para acabar este suplemento os comentamos cada uno de los juegos ganadores del concurso e insertamos el principio de su código que, no os olvidéis podéis encontrar entero en el CD-Rom si abrís los ficheros pertinentes. Seguro que la visión y el estudio de estos códigos os ayudará a mejorar algunos aspectos de vuestros propios trabajos de programación.

Por supuesto, nuestro concurso sigue en pie con los mismos premios. Ya sabéis: 25.000 pesetas para el ganador y 20.000 para los otros dos juegos elegidos. Esperamos con impaciencia recibir vuestros juegos. Ya sabéis dónde mandarlos, pero por si acaso hay algún despistado, os recordamos que podéis hacerlo por e-mail, correo normal o incluso entregarlos a mano en la dirección de la redacción:



Divmanía
C/ Alfonso Gómez, 42
Nave 1-1-2
28037, Madrid, España





Destacamos

En nuestro CD de portada incluimos los juegos que han resultado ganadores en este número y, cómo no, sus respectivos códigos para que veáis todo el proceso de creación. Estos son los afortunados:

- Aselo, un excelente matamarcianos con cierto aire "retro".
- Men In Green, un arcade 3D inspirado en juegos como Doom y Quake.
- Numbers, usa los guarismos para saltar de plataforma en plataforma.

Los Arrays

A la hora de representar la información tal y como se presenta en el mundo real, no es suficiente con disponer de elementos básicos de datos, sobre todo cuando esta información se presenta organizada y relacionada. A través de las estructuras de datos es posible no sólo la representación sino también la simplificación del tratamiento de elementos de datos que en la realidad aparecen relacionados.

Los tipo básicos presentados hasta ahora permiten representar elementos sin ninguna relación. Si se piensa en un programa como en un proceso que intenta representar una parte del mundo real y que éste tiene asociada una gran complejidad, es evidente que los lenguajes de programación deberán contar con elementos adecuados para tratarla.

En particular, existe un gran número de situaciones en las que la información se presenta organizada o le es aplicable una organización, de tal forma que el lenguaje deberá permitir al programador crear estructuras que reflejen dicha organización. A través de los tipos de datos estructurados, esto es posible.

Un tipo de datos estructurado no es más que un conjunto de datos relacionados, accesibles a través de un único nombre, a los que están asociados una serie de operaciones básicas. Los tipos de datos estructurados pueden clasificarse en estáticos y dinámicos dependiendo de su ciclo de vida. Éstos serán estáticos cuando su tamaño se fije en tiempo de compilación, es decir, que una vez declarados no se les permite crecer o disminuir a lo largo de la ejecución del programa. A diferencia de éstos, los tipos de datos dinámicos durante la ejecución del programa pueden ver aumentado o disminuido su número de elementos o componentes.

ARRAYS

Si se busca la organización que subyace en un horario de actividades, puede verse que ésta no es más que una tabla en la que se puede insertar en cada uno de sus "cuadros" una actividad para un día de la semana y una hora determinada. Cuando se desea consultar qué actividad se realizará en un momento determinado, basta con indicar el día de la semana y la hora, accediendo al cuadrado en el que se encuentra la actividad. Por lo tanto, además de la información que almacena un horario de actividades, es importante la información a través de la cual se conoce qué elemento se desea recuperar.

El anterior ejemplo permite introducir de forma natural el concepto de array, que no es más que una estructura que permite agrupar un conjunto de elementos del mismo tipo, denominado tipo base, pudiendo ser accesibles cada uno de los elementos agrupados por medio de un índice. Además, es posible referenciar a la estructura en su conjunto mediante el nombre que se le haya asignado. Volviendo al ejemplo inicial, el conjunto de actividades no sería más que el tipo base (por ejemplo, una cadena de caracteres), en el que cada actividad representa uno de los elementos. Mientras tanto, el tipo índice estaría compuesto por dos valores: el día de la semana y la hora. Además, la referencia a todos los elementos se podría realizar mediante un nombre genérico: horario de actividades (en la figura 1 se muestra una representación del ejemplo).

A partir del índice es posible establecer una ordenación de los elementos que componen un array. Así, la actividad que hay asignada en el horario de actividades los martes a las once precede a la actividad asignada el miércoles a las nueve. En la mayoría de las ocasiones esta característica será aprovechada, en particular cuando el orden en que se almacenen los elementos del array sea significativo.

Otra característica interesante de los arrays reside en que el índice es calculable, lo que permite asociar un significado al hecho de que un determinado elemento esté ocupando una posición determinada del array, así como facilitar el acceso a un conjunto de elementos que deban ser tratados de forma similar.

En particular, todos los tipos enumerados pueden ser parte del índice de un array. Ejemplos de estos tipos son los siguientes: entero, lógico, carácter, byte y word. Se puede definir la dimensión de un array como el número de elementos necesarios para poder acceder a cada uno de sus componentes. En caso de que se desee representar la tabla horaria mediante un array, se deberá dotar a éste de dos dimensiones, especificando un índice para la fila y otro para la columna. Cada lenguaje de programación limita la dimensión, aunque como mínimo se permiten dos. La utilización de arrays de más de tres dimensiones (si es que el lenguaje de programación lo permite) dificulta la comprensión de la estructura que lo representará, por lo que es aconsejable buscar una representación alternativa.

DECLARACION

La declaración de un array en cualquier lenguaje permite establecer el tipo de elementos que será capaz de almacenar y su número de componentes, pues se trata de un tipo de datos estático. En el caso particular de Pascal, resulta posible definir un array como

Horario de actividades

Índice

	Lunes	Martes	Miércoles	Jueves	Viernes
9	Física	Cálculo	Álgebra		Álgebra
10	Cálculo	Descanso	Cálculo	Álgebra	Cálculo
11	Álgebra		Física		Física

Índice

Componente
(10, Miércoles)
Valor "Cálculo"

FIGURA 1.

```
Const

 max_num_lista= 25 ;
 max_num_tabla= 100 ;
type

 rango_tabla = 1..max_num_tabla ;
 lista =array [1..max_num_lista] of boolean;
 tablero = array [rango_tabla, rango_tabla] of char ;

var
 tab1, tab2 : tablero;
 lista_aux : lista  ;
 var_arr : array[1..25] of real ;
```

Se han definido dos tipos array, uno unidimensional y otro bidimensional

Se han declarado tres arrays, dos de tipo tablero y uno de tipo lista

Declaración de una variable array sin definición previa de tipo

FIGURA 2.

variable (en la sección var) o como tipo(en la sección type). Es recomendable que si el array representa algún objeto con significado especial para el programa, aquel sea declarado como tipo y posteriormente se declaren tantas variables de dicho tipo como si de uno predefinido se tratara.

Para declarar un tipo de array se debe especificar un nombre de tipo y a continuación un igual, seguido de la palabra reservada array. A continuación, y tras abrir un corchete, se especificará un rango de valores que indicarán el límite superior e inferior del índice. En este punto puede cerrarse el corchete anteriormente abierto si tan sólo se desea disponer de un índice. Si por el contrario se desea declarar un tipo de array con más de un índice (multidimensional), entonces se procede a poner una coma y de nuevo a indicar límites superior e inferior del nuevo índice. Tras cerrar el corchete, se especifica tras introducir la palabra reservada of el tipo de los elementos del array. Una vez declarado el tipo de array, es posible declarar variables de dicho tipo cuando se considere conveniente. Si lo que se desea es declarar variables que sean arrays, el procedimiento es similar al anterior, pero debe producirse en lugares del programa en los que se puedan declarar variables, sustituyendo el símbolo igual "=" por el carácter dos puntos ":".

En la figura 2 puede verse la declaración de varios tipos y variables arrays. En primer lugar se han declarado en la sección const dos

```
Const
 max_columna = 10 ;
 max_fila = 15 ;
type
 columna = array [1..max_columna] of char ;
 tablero = array[1..max_fila] of columna ;
var
 t : tablero ;
 i, j : integer ;
 c : columna ;
begin
 ...
c := t[5] ;
for i := 1 to max_fila
   do
    begin
     for j := 1 to max_columna
        do
         write(t[i][j]) ;
     writeln ;
    end ;
 ...
end.
```

La componente de la variable t es de tipo columna (array)

Acceso a un elemento de un array bidimensional

FIGURA 4.

constantes con el número de elementos que permitirán almacenar diferentes tipos de arrays. Esta práctica resulta muy recomendable, pues si tras desarrollar el programa no se estimó correctamente la dimensión, es posible modificar estos valores y volver a compilar el programa. En la sección type se ha declarado un tipo subrango denominado rango_tabla, que permitirá por un lado declarar variables de dicho tipo, por lo que se asegura que cuando se acceda al array mediante una variable de dicho tipo no se accederá a elementos fuera del rango. Además, dicho tipo sirve para la construcción del tipo array bi-dimensional tablero, indicando el número de elementos de los que dispondrá, siendo en este caso un array con 10.000 elementos de tipo Boolean.

Hay que considerar la dimensión del array a la hora de tener acceso a sus componentes

A diferencia del anterior, en el tipo lista no se ha utilizado este método de declaración, disponiendo así el valor inferior y superior que puede tomar el índice de las variables definidas como lista. Por último, en la sección dedicada a la declaración de variables se han declarado variables de los anteriores tipos y una variable de tipo array sin definición previa de tipo. Es aconsejable declarar un tipo de array y posteriormente definir variables de dicho tipo a la manera en que se hizo con tablero, pues permite autodocumentar mejor el programa y definir un interfaz más claro entre los procedimientos y funciones que tengan parámetros de dicho tipo. Además, si el lenguaje de programación presenta una comprobación fuerte de tipos, es posible

```
Const
 max_lista = 25 ;
type
 rango_lista = 1..max_lista ;
 lista = array[1..max_lista] of real ;
 tablero = array[1..15, 1..10] of char ;
var
 l : lista ;
 t : tablero ;
 i_l : rango_lista ;
 i, j : integer ;
 total : real ;
begin
 ...
 total := 0 ;
 for i_l := 1 to max_lista
    do
     total := total + l[i_l] ;
 for i := 1 to 15
    do
     begin
      for j := 1 to 10
         do
          write(t[i,j]) ;
      writeln ;
     end ;
 ...
end.
```

Acceso a un elemento de un array unidimensional

Acceso a un elemento de un array bidimensional

FIGURA 3.

detectar en tiempo de compilación errores que en otro caso no se detectarían hasta el momento de la ejecución.

ACCESO A COMPONENTES

Aunque es posible acceder a la totalidad de una variable de tipo array utilizando el identificador de la variable, conviene comprobar si dos arrays son iguales mediante el símbolo "=" y asignar una variable de tipo array a otra de igual tipo. Esta clase de operaciones no son las habituales, ya que la mayoría de las operaciones se realizan sobre componentes individuales.

Para poder acceder a cada uno de los elementos de un array es necesario definir variables compatibles con el rango que se ha asociado al array (todos los lenguajes permiten, al menos, el tipo entero en sus diferentes versiones) como índice para el acceso a los componentes de un array.

EN TURBO PASCAL, LAS FUNCIONES Y PROCEDIMIENTOS QUE SE UTILIZAN PARA MANIPULAR CADENAS SON LAS SIGUIENTES:

- length (cadena): esta función devuelve la longitud lógica de la cadena. Es posible utilizar la expresión ord (cadena[0]) en sustitución de la función, pues en dicha posición se almacena en forma de byte la longitud de la cadena.
- Concat (cadena1, cadena2,...): esta función devuelve la cadena que resulta de concatenar las cadenas especificadas. Es posible utilizar el operador "+" para obtener el mismo resultado.
- Delete (cadena, pos, num): este procedimiento elimina de la cadena los num caracteres que se encuentren a partir de la posición pos.
- Pos (subc, cadena): esta función devuelve la posición a partir de la cual la cadena cad tiene como subcadena a subc. Si no es subcadena, devuelve el valor 0.
- Copy (cad, pos, num): esta función devuelve la subcadena de cad que se encuentra en la posición pos y contiene num elementos.
- Str (num, cadena): este procedimiento convierte el valor numérico representado por num en una cadena de caracteres.
- Val (cadena, num, codigo): este procedimiento convierte la cadena en un valor numérico,

Funciones en el Lenguaje C

Este capítulo va a dedicarse a otro tema muy importante d la programacíon bajo C: el uso de las funciones, las cuales, podemos decir que son el pilar fundamental sobre el cual s construyen todos los programas en C. Saber manejarlas perfectamente es algo que puede considerarse básico.

Una persona no puede hacer todas las tareas que se le presentan sin ayuda de otras personas. Se acude a un técnico para que se repare el televisor, se paga a alguien para que corte el césped, etcétera.

Los programas de ordenador están en la misma situación; no pueden manejar solos todas las tareas. Tienen que llamar a otras entidades de programación, denominadas funciones en lenguaje C, para realizar tareas específicas. En este artículo se explorará el tema de las funciones. Se verán una gran variedad de funciones, empezando con los casos más simples y dando los ejemplos que muestren algunas de las más usadas y útiles funciones de C.

¿QUÉ HACE UNA FUNCIÓN?

Una función en C tiene un propósito similar al que tiene una subrutina en BASIC y las funciones y procedimientos en Pascal. A continuación, se detalla el tema con de forma más explicativa.

EVITAR LA REPETICIÓN INNECESARIA DE CÓDIGO

Seguramente, la razón original de la existencia de las funciones (o subrutinas, como se las conoció primero) fue evitar escribir el mismo código una y otra vez. Suponga que tiene una sección de código en su programa que calcula la raíz cuadrada de un número.

Si, más adelante, en el programa quiere calcular la raíz cuadrada de un número diferente, sería demasiado tedioso tener que escribir de nuevo las mismas instrucciones. En su lugar, preferiría saltar a la sección del código que calcula las raíces cuadradas y volver de nuevo, una vez realizado el cálculo, al flujo normal del programa.

ORGANIZACIÓN DEL PROGRAMA

Esto es todo lo que hacían las subrutinas, y lo que todavía hacen en BASIC. Sin embargo, con el paso de los años se encontró que el uso de la idea de las subrutinas ayudaba a organizar los programas y que simplificaba su verificación. Si el programa se pudiera dividir en varias actividades separadas, y se pudiera asignar cada una de ellas a una subrutina, entonces cada subrutina podría ser escrita y comprobada de forma más o menos independiente. Separando el código en funciones modulares, se facilita el diseño y la comprensión de los programas.

INDEPENDENCIA

Esta idea toma consistencia, y se vio claramente que realizar subrutinas independientes del programa principal e independientes entre sí era una gran ventaja.

Las funciones fueron diseñadas para no escribir código repetido en los programas

Por ejemplo, este tipo de subrutinas tendría sus propias variables privadas; es decir, estas variables no podrían ser accesibles desde el programa principal o desde otras subrutinas. Esto significaba que un programador no necesita preocuparse por el uso accidental de los mismos nombres de variable en diferentes subrutinas; las variables de cada subrutina estarían protegidas de las intromisiones inesperadas desde otras subrutinas. Así, era sencillo escribir programas largos y complicados. Pascal y otros lenguajes de programación más modernos utilizan esta independencia, y el C también lo hace.

FUNCIONES SIMPLES

Como ya se ha comentado, emplear una función es como pagar a alguien para que realice una determinada tarea por nosotros. A veces, la ejecución con la persona es muy sencilla, a veces es más compleja.

Suponga que tiene una tarea que siempre se realiza exactamente de la misma forma, por ejemplo, recortar el césped. Cuando usted quiera hacerlo, llamará por teléfono a la persona encargada y le dirá, "es el momento, hazlo ahora". No necesita dar instrucciones, ya que esa tarea es lo único que tiene que hacer esta persona. No es necesario que le avise cuando el trabajo esté acabado. Usted supo que cortará el césped de la forma habitual, la persona lo hace, y eso es todo.

Veamos una función simple en el C que traba de la misma forma. En realidad, veremos dos cosas: un programa que llama o activa la función (equivalente a la acción de llamar po teléfono a la persona de cortar el césped) y la función en sí misma. He aquí un programa:

```
/* cajatext.c */
/* coloca un recuadro alrededor de un texto */
void linea (void);

main
{
  linea( );
  printf("\xDB TEXTO EN CAJA \xDB\n");
  linea( );
}

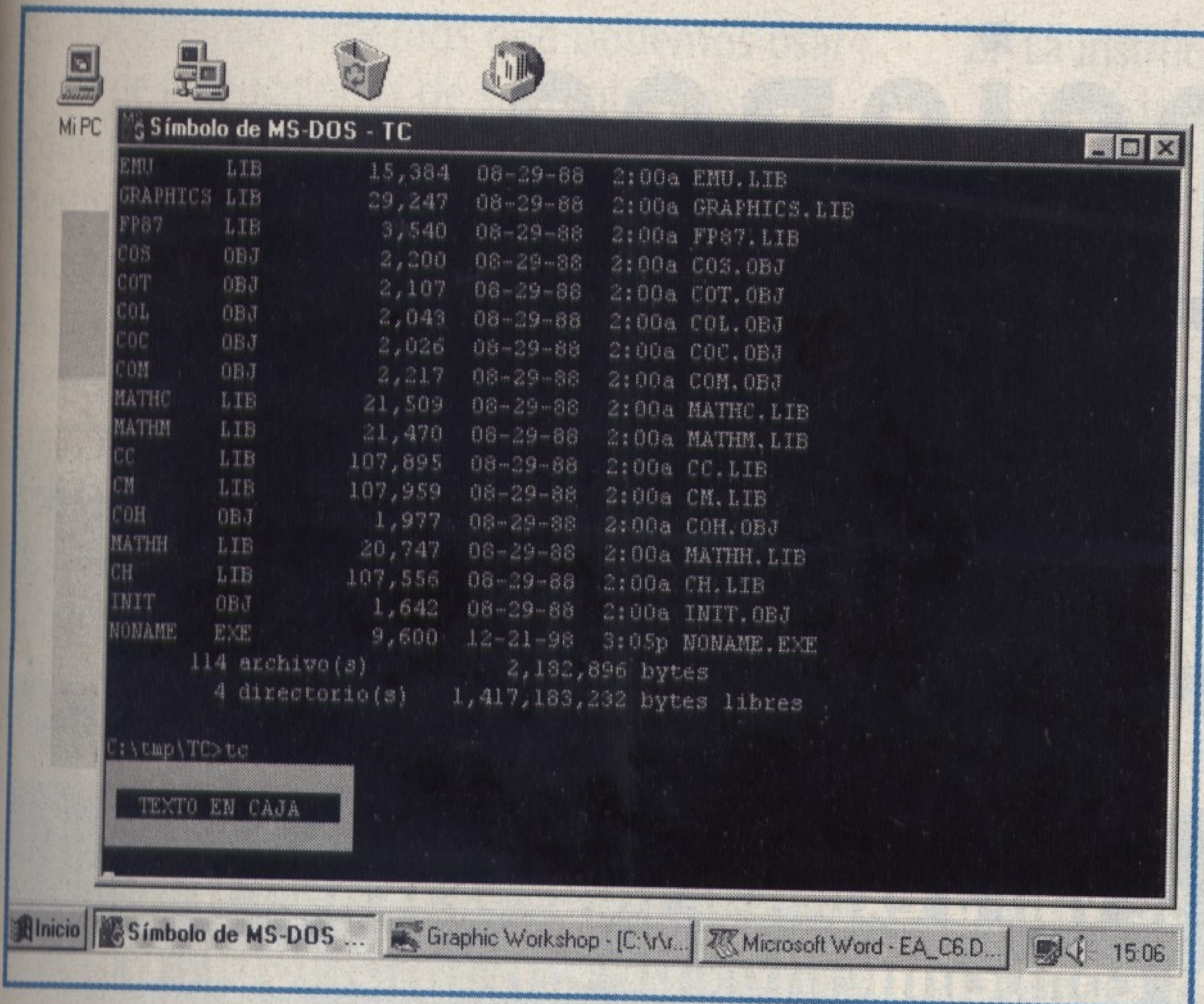
void linea (void)
{
  int j;
  for(j=1; j<=20; j++)
    printf("\xDB");

  printf("\n");
}
```

Este programa dibuja una caja alrededor del texto "Texto en caja": se dibuja una línea de rectángulos a lo largo de la pantalla (utilizando el carácter gráfico "\xDB"). Luego se imprime el nombre del emperador, precedido y terminado por un rectángulo, para así formar los extremos, y finalmente se dibuja otra línea. En lugar de escribir el código para dibujar la línea dos veces, hemos construido una función, denominada *linea.*

ESTRUCTURA DE LAS FUNCIONES

El programa *cajatext.c* parece estar formado por dos programas. En realidad, contiene dos



RESULTADO DE EJECUTAR EL PRIMER EJEMPLO DEL ARTÍCULO, DONDE APARECE EL TEXTO 'TEXTO EN CAJA'.

funciones: la primera se llama *main( )* y la segunda se denomina *linea( )*. Como vimos en el capítulo 1, *main( )* es una función, y por tanto, no es sorprendente que la función *linea( )* se le parezca bastante. La única característica especial acerca de *main( )* es que no sea la primera función en la lista; podrá colocar otras funciones por delante de *main( )*, sin embargo, éste siempre se ejecuta primero.

En este ejemplo, *main( )* llama a la función *linea( )*. "Llama" significa que causa su ejecución; *main( )* llama dos veces a k para dibujar las líneas inferior y superior de la caja. Al emplear una función, se ven involucrados tres elementos del programa: la definición de la función, la llamada a la función y el prototipo de la función. Veamos ordenadamente cada una de ellas.

• *Difinición de la función*: se denomina definición de la función a la función propiamente dicha. La definición empieza con una línea que incluye el nombre de la función, entre otros elementos.

```
void linea (void)
```

Esta línea es el declarador. El void inicial significa que *linea( )* no devuelve nada, y el segundo significa que no tiene argumentos. Veremos el retorno de valores y los argumentos más adelante en este capítulo; pero ahora, asegúrese sólo de incluir esos void. Ambos son necesarios.

Vea que el declarador no termina con punto y coma. No es una instrucción de programa cuya ejecución hace que ocurra algo. Más bien, le dice al compilador que en ese lugar empieza la definición de una función.

La definición de la función continúa con el cuerpo de la función; es decir, las líneas de código que realizan el trabajo. La figura 5.3 muestra el declarador y el cuerpo de tal función, que constituyen la definición de la función.

```
void linea(void)
{
 int j;
 for(j=1; j<=20; j++)
   printf("\xDB");
 printf("\n");
}
```

El cuerpo de la función está entre llaves. Normalmente, esas llaves están colocadas en el margen izquierdo.

• *Llamada a la función*: desde *main( )* se llama a la función *linea( )*, escrita por el usuario, del mismo modo que se hace con el resto de las funciones de la librería del C, por ejemplo: *printf( )* y *getche( )*. La llamada se realiza utilizando el nombre de la función seguido por ambos paréntesis. Éstos son necesarios para que el compilador conozca que se está refiriendo a una función y no a una variable o algo similar. La llamada a una función constituye una instrucción C, por tanto, finaliza con un punto y coma.

```
linea( );
```

La llamada a la función transfiere el control del programa al código existente en la definición de *linea( )*. La función dibuja su fila de cuadrados sobre la pantalla, y devuelve el control a *main( )*, en la instrucción siguiente a la que llamó a la función.

Una función se declara de la misma manera al principio del programa

• Prototipo de una función: éste es el tercer elemento relacionado con la función. Está formado por unas líneas antes del comienzo de *main( )*.

```
void linea(void);
```

¿CUÁL ES SU PROPÓSITO?

Ya ha visto muchos ejemplos de variables en los programas de C. Todas las variables se declaran por su nombre y el tipo de dato al principio de la función donde se van a utilizar. Una función se declara de la misma manera al principio del programa. La declaración de la función (o prototipo, ambos términos significan lo mismo) le dice al compilador el nombre de la función, el tipo de dato que devuelve la función (si hay alguno) y el número y el tipo de dato de los argumentos de la función (si los hubiera). En este caso, la función no devuelve nada ni requiere ningún argumento.

Lo que hay que recordar sobre el prototipo es que los tipos de datos (en este ejemplo ambos son void), declarados tanto en el prototipo de la función como en su parte declarativa, deben coincidir. En caso contrario, el compilador nos devolverá un mensaje de error y no finalizará la compilación.

UN EJEMPLO SONORO

En este caso se utiliza el carácter especial *\x7*, denominado *Bell* (campana) en el código ASCII. Al mandar este carácter a un IBM, en lugar de sonar una campana, sonará un pitido. Veamos el programa:

```
/* sonido */
/* comprueba la función dosbeep */
void dosbeep(void);

main( )
{
dosbeep( );
printf("Escriba algún carácter");
getche( );
dosbeep( );
}

/* definición de la función dosbeep */
/* hace sonar el altavoz dos veces */
void dosbeep(void)
{
 long j;
printf("\x7");
for(j=1; j<100000; j++)
   ;
print("\x7");
}
```

Este programa llama primero a la subrutina *dosbeep( )*, que hace lo que indica su nombre: suenan dos *beeps* separados por un corto intervalo de tiempo. El programa requiere luego que se pulse una tecla; una vez realizada esta operación, suenan de nuevo los dos *beeps*.

Observe cómo se ha construido la pausa: de ha definido un bucle for que se ejecuta 100.000 veces. Sin embargo, no existen instrucciones en el cuerpo del programa. En su lugar, hay una instrucción que consta de un punto y coma. Esto constituye una instrucción vacía: instrucción que no consiste en nada. Su único cometido es acabar el bucle for.

Variables, instrucciones de salto y procedimientos

Ahora que han quedado explicados ya todos los conceptos que se pueden considerar básicos, se pasará a la ampliación de nuestros conocimientos.
Primeramente, se hará un repaso de la descripción de lo que es una variable propiamente dicha, ya que es un concepto que a muchos puede que les resulte difícil de entender, y que es muy importante para poder realizar correctamente punteros de memoria y manejarlos.

En este artículo se van a explicar nuevos temas que ampliarán nuestros conocimientos de ensamblador: tras una breve descripción de lo que es una variable (para los que aún están despistados), se pasa ya a detallar todas las instrucciones de salto que existen en ensamblador y para acabar, se profundiza un poco en otros temas como son los procedimientos o las banderas de estado.

Una variable no es más que un número de bits a los que se les da un nombre

Una vez hecho esto, se explica todo lo relacionado con las instrucciones de salto, incluyendo tanto las llamadas condicionales (las equivalentes a las *if/then/else* del lenguaje C o BASIC), como las incondicionales, que son en general las instrucciones más importantes para cualquier programador, sea cual sea el lenguaje que usa y el área de la programación en que se mueve.
Para acabar, se pasará a detallar más sobre el funcionamiento y manejo de los procedimientos, y de cómo aprovechar la capacidad de muchas instrucciones para modificar las banderas de estado (del famoso registro de estado).

REALIDAD DE UNA VARIABLE

Para entender mejor qué es una variable, decir que cuando declaramos una variable en un programa, lo que realmente estamos haciendo es guardar una zona de memoria (de tamaño dependiente del tipo de variable), que estará destinada a contener información útil para el programa.
Dicho esto, es fácil deducir que una variable no es más que un puntero a memoria que referencia al primer bit donde hemos guardado su contenido, y que está formado, cómo no , por un segmento base (bloque donde se definió la variable) y por un OFFSET que corresponde al número de bit donde se encuentra contando como base cero al bit donde empieza el segmento, y así, en el ejemplo de querer sumar 5 a una variable llamada COCHE , que sería ADD COCHE,5 , lo que estamos haciendo realmente es sumar 5 al contenido de una dirección de memoria que contiene el valor que hemos asignado a COCHE , lo cual quedaría realmente por poner un ejemplo: *ADD word ptr DS:[5000],5.*
El hecho de dar un nombre a la zona de memoria asignada para contener una información, se hace para que sea más fácil programar y para ahorrarnos tener que indicar cada vez a qué tipo de variable apunta dicho puntero. Así, por ejemplo, las del primer fuente *VARIABLE1*, *VARIABLE2* y *RESULTADO* , no son más que tres direcciones de memoria (*OFFSETS*) relativas a una base (en el ejemplo el segmento *DATOS*) y la línea *MOV AX,VARIABLE1* realmente no es más que un *MOV AX,DS:[00]*. el cual, como podemos comprobar, no es más que una instrucción que usa el modo de direccionamiento 3.1 explicado en el anterior capítulo. De la conversión de nombres (*ETIQUETAS*) a direcciones de memoria, se encarga el ensamblador cuando convierte nuestros fuentes a extensión OBJ, antes de que se conviertan a EXE con el *LINK.EXE*.

Símbolo de MS-DOS - pwbed
File Edit View Search Make Run Options Browse Help
Help: Declare Variables
◄Example► ◄Up► ◄Contents► ◄Index► ◄Back►

Type	Abbr	Size	Integer Range	Types Allowed
BYTE	DB	1 byte	-128 to +255	Character, String
WORD	DW	2 bytes	-32,768 to +65,535	16-bit near ptr, 2 characters, double-byte character
DWORD	DD	4 bytes	-2Gig to +4Gig-1	16-bit far ptr, 32-bit near ptr, 32-bit long word
FWORD	DF	6 bytes	—	32-bit far pointer
QWORD	DQ	8 bytes	—	64-bit long word
TBYTE	DT	10 bytes	—	BCD, 10-byte binary numbers

File Operations Menu | pseudo | 00022.003

PARA REPASAR LAS VARIABLES, EL PROPIO MACROENSAMBLADOR TRAE DETALLADAS EXPLICACIONES DE LOS TIPOS Y LA FORMA DE DEFINIR CADA UNA.

LAS ETIQUETAS

Ahora vamos pasar a otro tema, el de las instrucciones de salto, pero antes vamos a recordar qué son las etiquetas.
Como pasa en muchos lenguajes como el BASIC, cada línea de código en ensamblador puede poseer un nombre o *etiqueta* que la identifique. Gracias a

esta posibilidad, tenemos que podemos usar el set de instrucciones de salto que posee el código máquina *escritas* en el micro destinadas al control de ejecución de los programas.
Antes de entrar en instrucciones, sólo recordar al lector que las etiquetas tienen unas normas y limitaciones de sintaxis, que son éstas:

1. Se definen como un nombre seguido de dos puntos (sin espacio en medio).
2. La instrucción de código asociada puede ir después de los dos puntos con un espacio en medio como mínimo, aunque también se pueden separar etiquetas e instrucción tantas líneas en blanco como se desee.
3. El nombre que la define puede poseer un máximo de 31 caracteres, que pueden ser tanto letras de la A hasta la Z, como números del 0 al 9 y caracteres raros, como son *Arroba*, *#*, *&*, etcétera.
4. Por último, recordar que no se diferencian mayúsculas de minúsculas, por lo que los programadores acostumbrados al C y C++ deberán tener cuidado de no duplicar etiquetas sin querer. Pasamos a detallar ejemplos de etiquetas.

LISTADO 1 CON LAS INSTRUCCIONES DE SALTO CONDICIONAL

- **JA: salta si superior sin considerar signo.**
- **JAE: salta si superior o igual (sin signo).**
- **JB: salta si menor sin considerar signo.**
- **JBE: salta si inferior o igual sin signo.**
- **JC: salta si hay acarreo (si CF=1).**
- **JCXZ: salta si CX es cero.**
- **JE: salta si iguales.**
- **JG: salta si mayor, considerando signo.**
- **JGE: salta si mayor o igual, considera signo.**
- **JL: salta si menor, considerando signo.**
- **JLE: salta si es menor o igual.**
- **JNA: salta si no es superior, sin signo.**
- **JNB: salta si no es menor, sin signo.**
- **JNBE: salta si no menor/igual sin signo.**
- **JNC: salta si no hay acarreo.**
- **JNE: salta si no son iguales.**
- **JNG: salta si no es mayor, no considera signo.**
- **JNGE: salta si no es mayor o igual, sin signo.**
- **JNL: salta si no es menor, con signo.**
- **JNLE: salta si no menor o igual (con signo).**
- **JNO: salta si desbordamiento.**
- **JNP: salta si no hay paridad.**
- **JNS: salta si no hay signo presente (positivo).**
- **JNZ: salta si no es cero.**
- **JO: salta si hay desbordamiento.**
- **JP: salta si hay paridad.**
- **JPE: salta si hay paridad par.**
- **JPO: salta si hay paridad impar.**
- **JS: salta si hay signo.**
- **JZ: salta si es cero.**

ESQUEMA DEL REGISTRO DE BANDERAS



ESQUEMA DEL MAPA INTERNO DEL REGISTRO DE BANDERAS.

LAS INSTRUCCIONES DE SALTO

En ensamblador, como en casi todos los lenguajes, existen dos tipos de saltos de ejecución. Por un lado tenemos las llamadas *instrucciones de salto condicionales* y por otro las *instrucciones de salto incondicional*.
Las de salto condicional son las que realizan el salto sólo en caso de producirse una situación determinada (condición). Estas condiciones en ensamblador siempre se representan por situaciones aritméticas, puesto que se basan en comparaciones numéricas del tipo *A>B*, *A=0*, *Si A=Negativo*, etcétera.
Las de salto incondicional, por otro lado, son muy simples de entender. Realizan el salto de ejecución a la línea indicada por la propia instrucción, sin tener en cuenta ninguna condición de estado de ejecución.
A continuación vamos a detallar el funcionamiento y uso de cada tipo.

SALTOS CONDICIONALES

En ensamblador, las condiciones son a grandes rasgos como en todos los lenguajes, aunque con la limitación de que no se pueden realizar comparaciones múltiples simultáneas. Por ejemplo, lo que en un lenguaje como el C sería tan fácil como la expresión *IF(vA>(vB & vC) función();* en ensamblador el programador lo tendría que dividir en 2 pasos (comparar *vA* con cada uno de forma encadenada).
Todas las instrucciones condicionales clásicas del ensamblador (ver *listado 2*), tienen la misma sintaxis básica, y se forman a partir de la *J* de *Jump* y de una o más siglas posteriores, que indican la condición a la que representan.
Después, sólo nos queda indicar la etiqueta destino hacia la que queremos saltar como parámetro de instrucción. Así, por ejemplo, para indicar que queremos realizar un salto a la etiqueta *LINEA_1* si se cumple que *A es mayor a B*, que en inglés sería *Jump if Above or Equal*, solamente tenemos que escribir *JAE LINEA_1*.
Además de esto, las comparaciones se pueden clasificar en condicionales con signo y sin signo, que usaremos según necesitemos considerar a los elementos de las comparación como números enteros con o sin signo. En programación ensamblador, por tradición, cuando se habla de mayor y menor (*great & less*) se están indicando números con signo, y al decir superior e inferior (*above & below*), se está hablando de números sin signo.

LISTADO 2, BITS DEL REG. FLAGS QUE ALMACENAN VALORES DE COMPARACION:

- **Bit 0-CF / Carry flag.**
- **Bit 2-PF / Parity flag.**
- **Bit 4-AF / Auxiliar flag.**
- **Bit 6-ZF / Zero flag.**
- **Bit 7-SF / Sign Flag: Si=1.**
- **Bit 11-OF / Overflow flag.**

Autor: Francisco Blázquez Munue

Aselo

El afortunado ganador de este número es un atractivo matamarcianos que nos retrotaerá a otros tiempos en los que las máquinas recreativas eran el pan nuestro de cada día. Además de su estupenda jugabilidad destaca por su atractivo gráfico y su completo menú de opciones. Nuestra enhorabuena para el autor.

Aquí está nuestro ganador de este mes. Y no creáis que ha sido fácil darle el premio. Los juegos que nos llegan cada vez tienen más calidad y son más numerosos. Nos ha llevado bastante tiempo decidirnos entre los que nos han llegado para este número. Otorgarles los merecidos premios nos ha costado incluso discusiones en la redacción. Y para que veáis

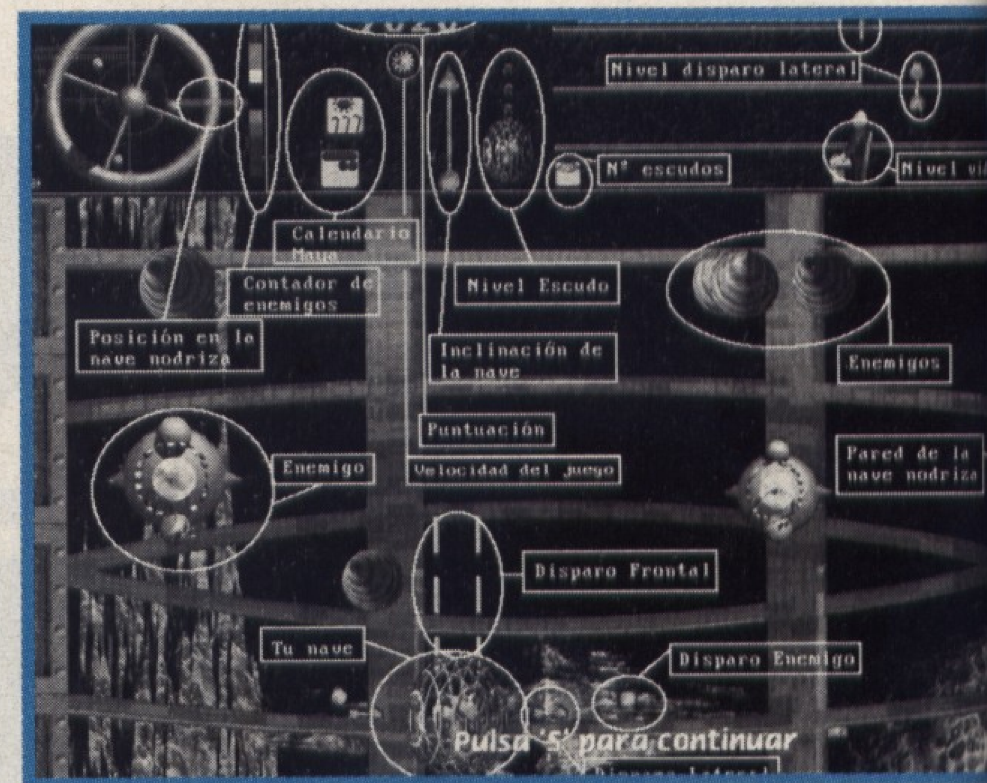
En el ocaso del siglo XX, el espíritu del hombre se encontraba inquieto

que no mentimos puede que, en algún número de Divmanía, incluyamos en el CD-Rom de la revista todos los juegos que nos han llegado a la redacción, los premiados y los no premiados. Los lectores que, seguro, han invertido meses, cuando no años, como ha ocurrido en algún caso, en su trabajo merecen que sus creaciones sean divulgadas de alguna forma. Además, seguro que todos los programadores aprenderán algo de sus aciertos y de sus errores.

Aselo no tiene nada que envidiar a los juegos de las antiguas máquinas recreativas matamarcianos en los que está inspirado, incluso se permite el lujo de mejorar gráficamente los juegos que cargaban estos viejos aparatos.

El argumento del juego reza de la siguiente guisa:

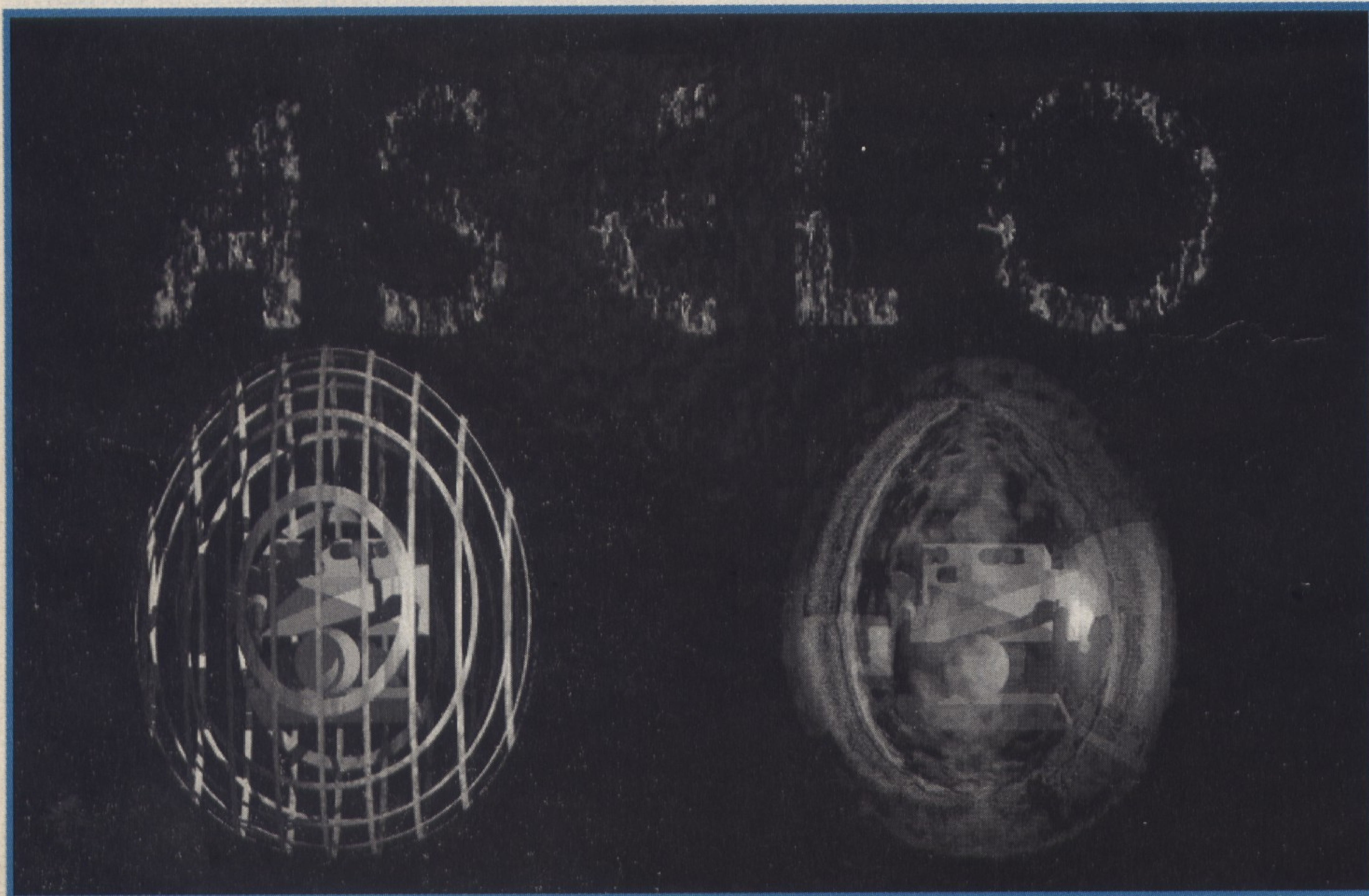
"En el ocaso del siglo XX, el espíritu del hombre se encontraba inquieto. Sabiendo que el sistema solar se quedaba pequeño, la desorientación hacía mella en una Humanidad que no encontraba el camino



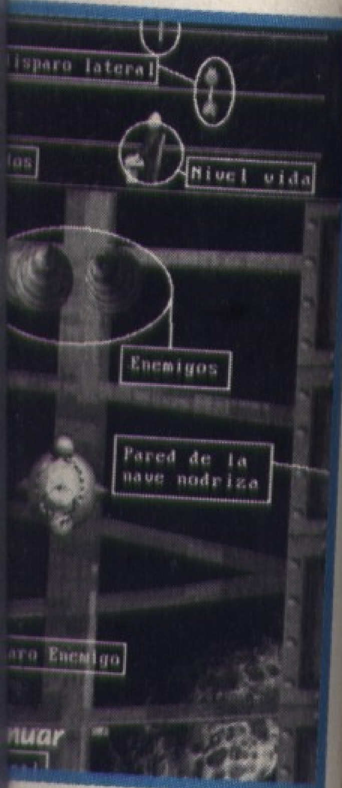
hacia nuevos mundos. La conciencia global..."

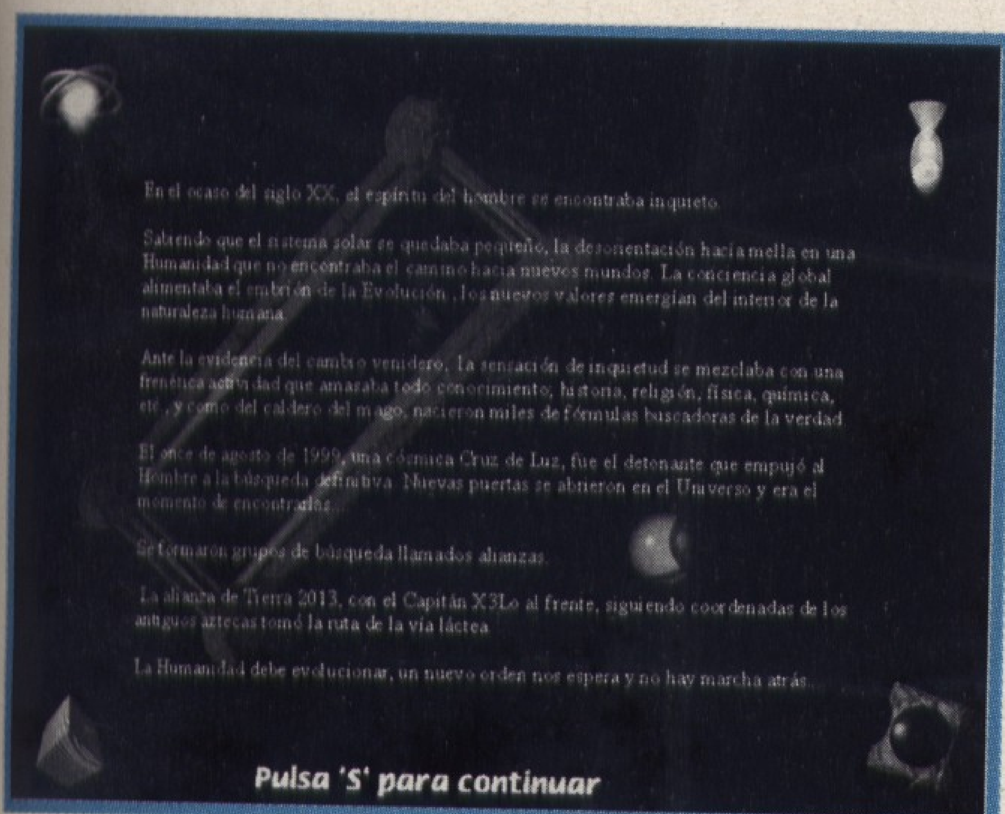
Encontraréis el resto si cargáis el juego en vuestro ordenador. Y aquí os damos un peda de código fuente que os servirá para saber como se ha realizado este juego:

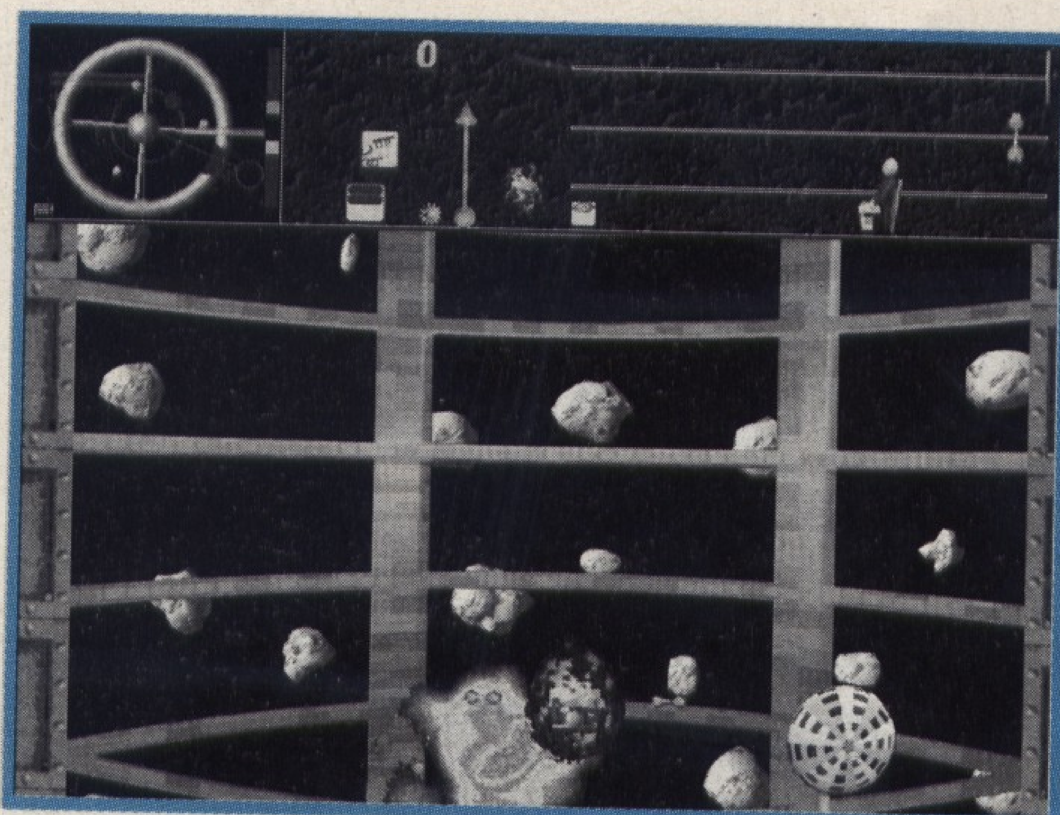
```
GLOBAL
  g0;G1;g2;g3;g4;g5;g6;g7;
  mx;
  my;
  vale;
  id0;
  id1;
  id2;
  id3;
  id4;
  Orientacion;
  alto;
  vel;
  SD;
  ST;
  tiempo2;
  burbujaUno [359];
  con_Burbujauno;
  alien [359];
  con_alien;
  helice [359];
  con_helice;
  son_laser;
  son_laser2;
  son_knock;
  son_tunel;
  son_explo1;
  son_muelle;
  son_vida;
  son_escu;
  son_final;
  son_arriba;
  son_lado;
  son_pared;
  son_ovni;
  son_el;
  son_fase1;
  son_fase2;
  son_fase3;
  son_fase4;
  son_esac;
```

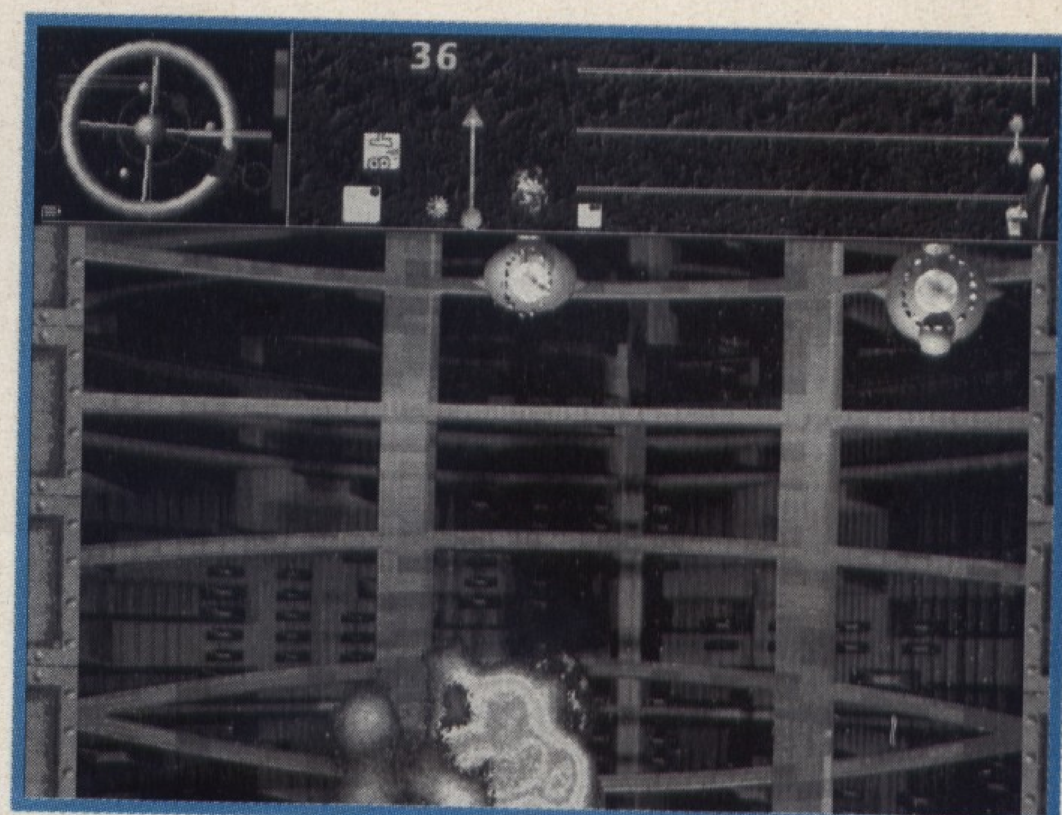


uez Munuera









encia

juego en
nos un pedazo
ara saber

```
vueltas;

heroe;
avx;
capa;
idolo;
dia;
mis;
mii;
ais;
aii;
marcos;
fue_malos;
prese;
explica;
explica2;
son_dadoporm;
barrarriba;
tipotiro;
fuerzabuelo;
fuerzanieto;
Nivel;
nescudo;
suelo;
puntos;
titulo;
Magia;

//*************************************
// INDICADORES DE APARICION DE
ENEMIGOS
//*************************************

GB1;
GB2;
GT1;
GT2;
GA1;
GA2;

N_ORI_CANGRE;        // A que altura
comienzan los ...
N_PROB_CANGRE;       // Con qué
probabilidad salen los...
N_NUM_CANGRE;        // Qué número de ....
N_ORI_ARANA;
N_PROB_ARANA;
N_NUM_ARANA;
```

```
N_ORI_TABLA;
N_PROB_TABLA;
N_NUM_TABLA;


N_AZAR_ENER;         // Probabilidad
aparición de Energía
N_AZAR_DLAT;         // ... disparo lateral
N_AZAR_DREC;         // ... disparo frontal
N_AZAR_ESCU;         // ... escudos

N_VIDA_CHTAB;        // VIDA QUE QUITA SI
DA EN..   Tablas
N_VIDA_CHARA;        // ... Arañas
N_VIDA_CHBUR;        // ... Burbujas
N_VIDA_CHGRAN;       // ... Gran enemigo
N_VIDA_PARED;        // ... paredes
N_VIDA_DIS3;         // ... Disparo enemigo

N_TOTAL_DLAT;        // NIVELES Totales
N_TOTAL_DREC;
N_TOTAL_VIDA;
N_IND_DLAT;          // Niveles parciales
N_IND_DREC;
N_IND_VIDA;
N_IND_ESCU;
N_IND_ESCUS;
                     // DISPAROS ENEMIGOS
N_FREC_TBUR;         // Frecuencia
N_VEL_DBUR;          // Velocidad
N_VEL_DARA;
N_VEL_DHAB;

N_FREC_ESCU;         // Caducidad de escudos


N_PUNTUACION;        // Puntos obtenidos


//--------------------------------------
--------

Opcion[6,4];         // Menú de opciones
lopcion;

Local
m;
```

```
//---------------------------------------
INICIO ---------------------------------

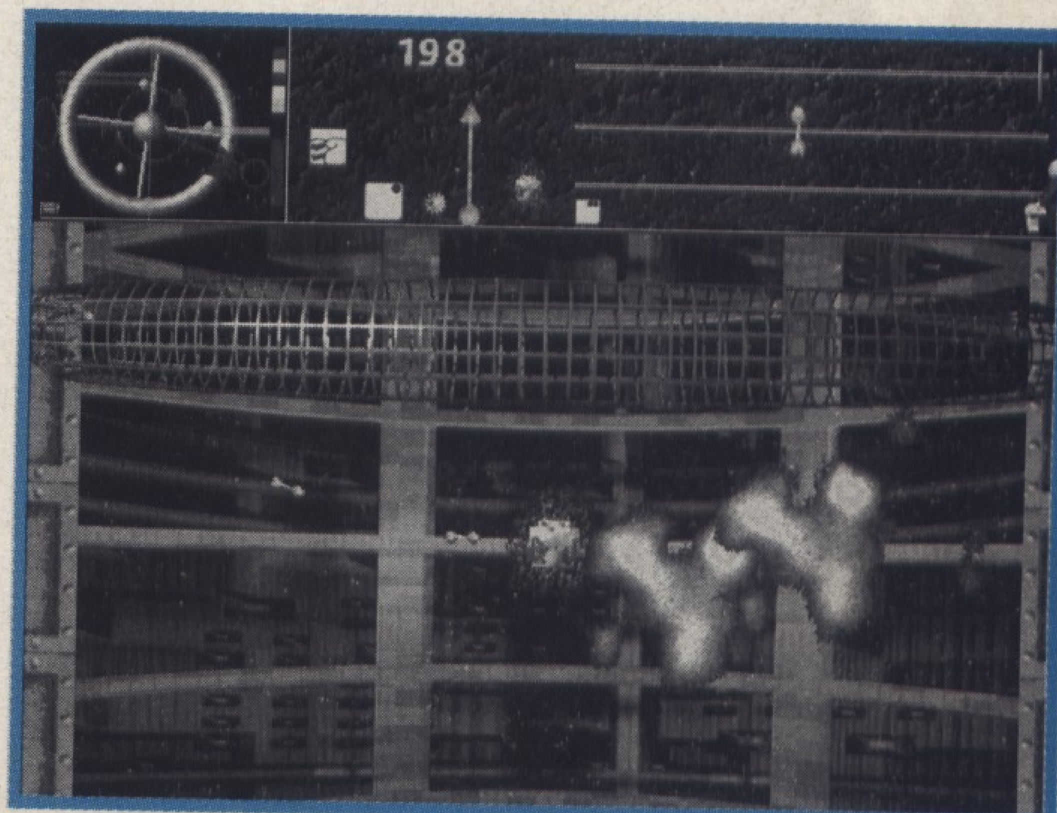
BEGIN

m=0;

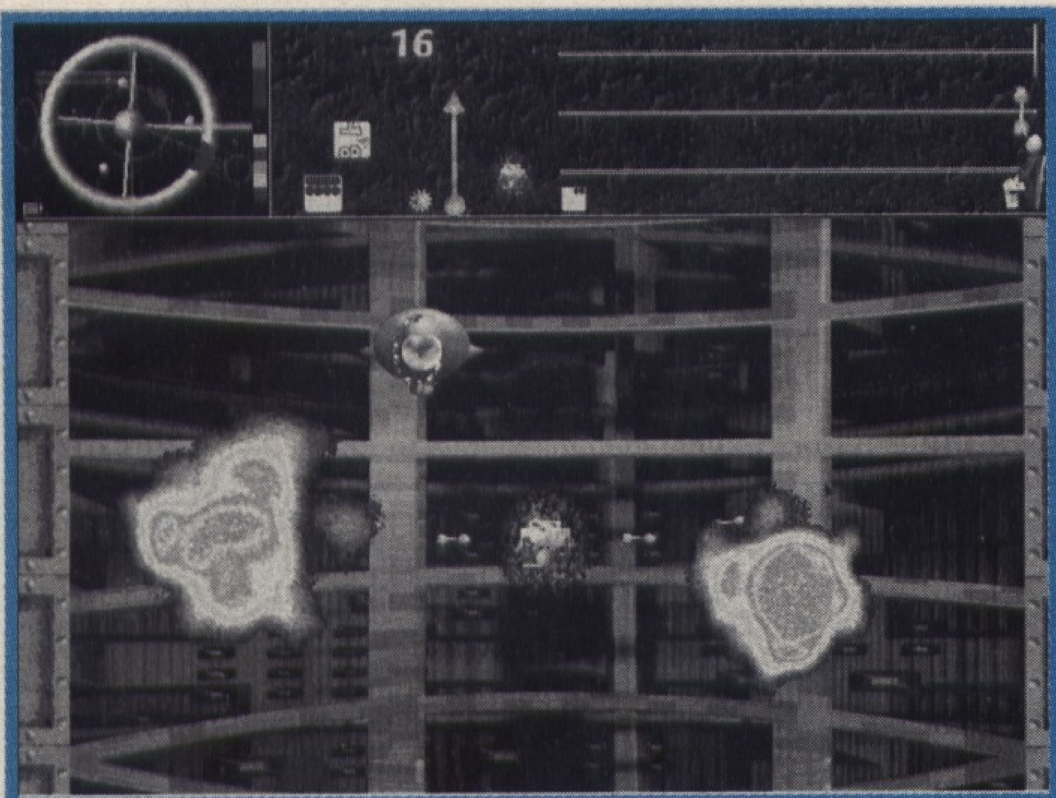
Opcion[0,0] = "F1.   Presentación";
Opcion[1,0] = "F2.   Instrucciones";
Opcion[2,0] = "F3.   Escenario:    En la
bodega interior";
Opcion[2,1] = "F3.   Escenario:    En las
costas de Panche-Be";
Opcion[2,2] = "F3.   Escenario:    Sobre las
ruinas de Itzaes";
Opcion[2,3] = "F3.   Escenario:    En el
cinturón Kinan";
Opcion[3,0] = "F4/F5. Velocidad:    ";
Opcion[4,0] = "F6.   Jugar";
Opcion[5,0] = "F7.   Calibrar velocidad";
Opcion[6,0] = "F10.  Salir";

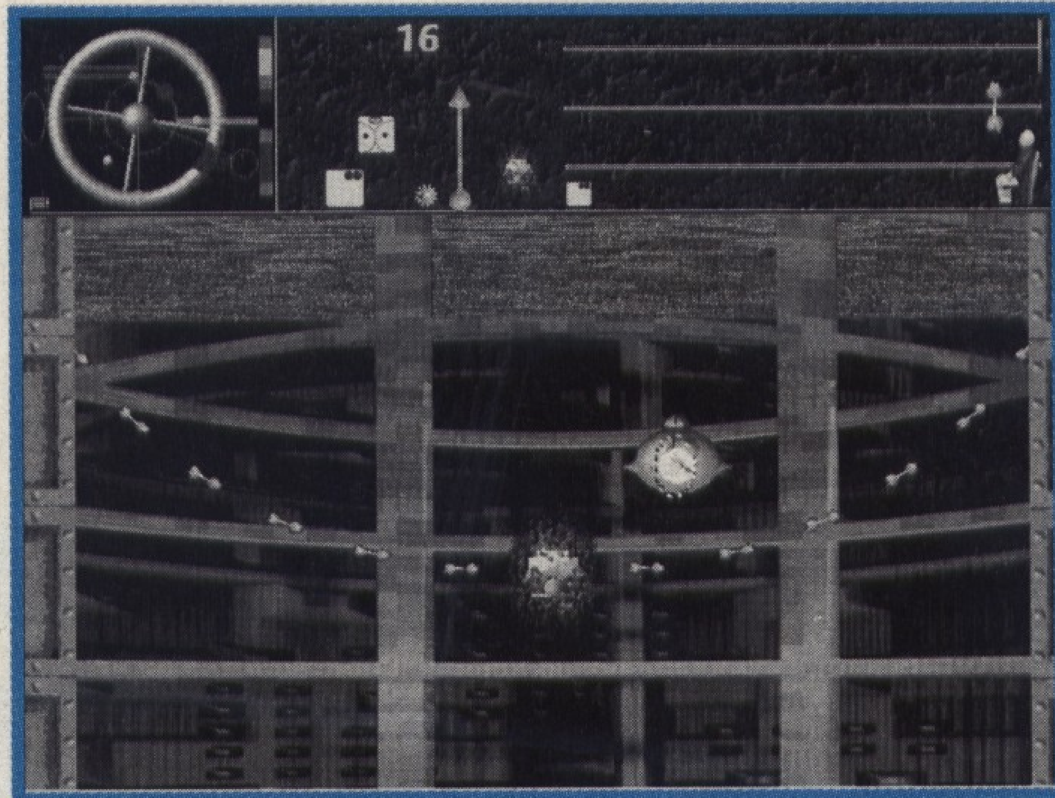

//***  Carga de ficheros para la aplicación
***

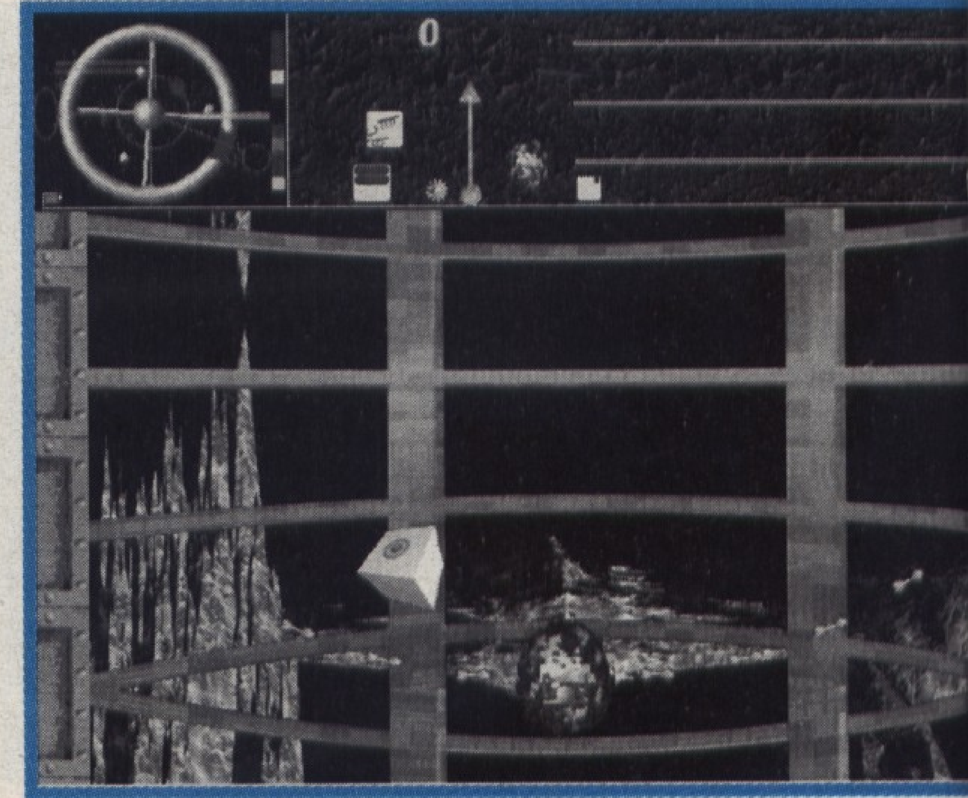

load_pal("\001.pal");
load_fpg("\001.fpg");
son_laser = load_pcm("\001.pcm",0);
son_dadoporm = load_wav("\002.wav",0);
son_explo1 = load_wav("\003.wav",0);
son_muelle = load_wav("\004.wav",0);
son_vida = load_pcm("\005.pcm",0);
son_Final = load_pcm("\006.pcm",0);
```



Juegos ganadores: 1º








```
son_arriba = load_wav("\007.wav",0);
son_lado = load_wav("\008.wav",0);
son_pared = load_wav("\009.wav",0);
son_escu = load_wav("\010.wav",0);
son_laser2 = load_wav("\011.wav",0);
son_knock = load_wav("\012.wav",0);
son_fase1 =load_wav("\013.wav",0);
son_fase2 =load_wav("\014.wav",0);
son_fase3 =load_wav("\015.wav",0);
son_fase4 =load_wav("\016.wav",0);
son_esac =load_wav("\017.wav",0);
son_tunel = load_wav("\018.wav",0);
son_ovni = load_wav("\018.wav",0);
explica= load_fnt("\001.FNT");
explica2= load_fnt("\002.FNT");
fue_malos= load_fnt("\003.FNT");
Puntos= load_fnt("\004.FNT");
Titulo=load_fnt("\005.FNT");


hola();   // Llama a pantalla de presentación

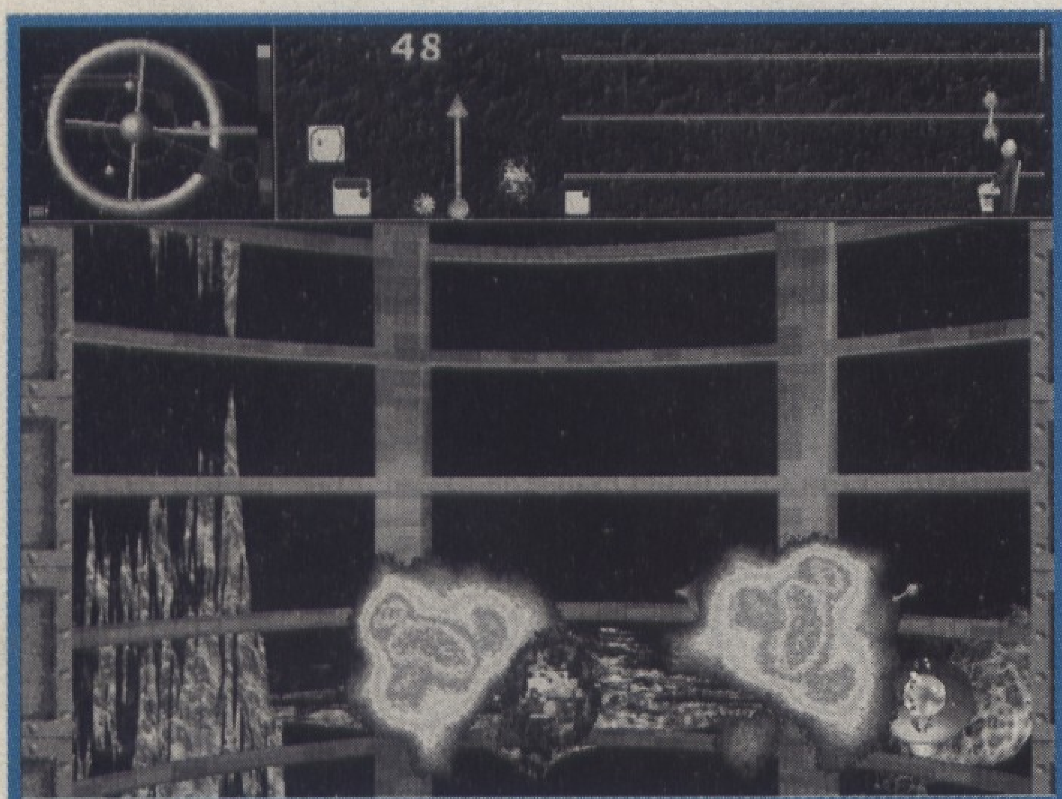
LOOP    // Básico bucle de acción.


nivel=0;
marcos=18;

menuOPciones();  // Muestra Menú

delete_text(all_text);
load_pal("\002.pal");

        //
```




```
********************************************
  switch (nivel)        // VALORES EN FUNCION
DEL NIVEL SELECCIONADO
                                          //
********************************************
  case 0;
   SD=2;
   ST=1;

                    // INDICADORES DE
APARICION DE ENEMIGOS

   GB1=120;
   GB2=147;
   GT1=150;
   GT2=179;
   GA1=327;
   GA2=356;
   sound(son_fase1,256,256);

   N_ORI_CANGRE=0;
   N_PROB_CANGRE=3;
   N_NUM_CANGRE=4;
   N_ORI_ARANA=200;
   N_PROB_ARANA=4;
   N_NUM_ARANA=3;
   N_ORI_TABLA=100;
   N_PROB_TABLA=3;
   N_NUM_TABLA=4;

   N_AZAR_ENER=9994;
   N_AZAR_DLAT=9992;
   N_AZAR_DREC=9992;
   N_AZAR_ESCU=9985;


   N_VIDA_CHTAB=50;   /// VIDA QUE QUITA SI
DA EN..
   N_VIDA_CHARA=60;
   N_VIDA_CHBUR=40;
   N_VIDA_CHGRAN=300;
   N_VIDA_PARED=50;
   N_VIDA_DIS3=25;

   N_TOTAL_DLAT=1000;    /// NIVELES
   N_TOTAL_DREC=1000;
   N_TOTAL_VIDA=1000;
   N_IND_DLAT=1000;
```

```
   N_IND_DREC=1000;
   N_IND_VIDA=1000;
   N_IND_ESCUS=2;
   N_IND_ESCU=0;
                    //// DISPAROS ENEMIGOS
   N_FREC_TBUR=30;
   N_VEL_DBUR=15;
   N_VEL_DARA=15;
   N_VEL_DHAB=10;

   N_FREC_ESCU=20;


  End


  case 1;
   SD=6;
   ST=5;

  // INDICADORES DE APARICION DE
ENEMIGOS

   GB1=267;
   GB2=296;
   GT1=357;
   GT2=386;
   GA1=327;
   GA2=356;
   sound(son_fase2,256,256);
   N_ORI_CANGRE=0;
   N_PROB_CANGRE=3;
   N_NUM_CANGRE=4;
   N_ORI_ARANA=190;
   N_PROB_ARANA=4;
   N_NUM_ARANA=3;
   N_ORI_TABLA=100;
   N_PROB_TABLA=3;
   N_NUM_TABLA=4;


   N_AZAR_ENER=9994;
   N_AZAR_DLAT=9994;
   N_AZAR_DREC=9994;
   N_AZAR_ESCU=9985;

   N_VIDA_CHTAB=050;   /// VIDA QUE QUITA
SI DA EN..
   N_VIDA_CHARA=065;
```

Juegos ganadores: 1º

```
N_VIDA_CHBUR=045;
N_VIDA_CHGRAN=400;
N_VIDA_PARED=60;
N_VIDA_DIS3=35;

N_TOTAL_DLAT=1000;    /// NIVELES
N_TOTAL_DREC=1000;
N_TOTAL_VIDA=1000;
N_IND_DLAT=1000;
N_IND_DREC=1000;
N_IND_VIDA=1000;
N_IND_ESCUS=2;
N_IND_ESCU=0;
           //// DISPAROS ENEMIGOS
N_FREC_TBUR=25;
N_VEL_DBUR=15;
N_VEL_DARA=20;
N_VEL_DHAB=15;

N_FREC_ESCU=20;

end
case 2;
 SD=2;
 ST=3;

// INDICADORES DE APARICION DE
ENEMIGOS

GB1=120;
GB2=149;
GT1=297;
GT2=326;
GA1=180;
GA2=209;
sound(son_fase3,256,256);
N_ORI_CANGRE=0;
N_PROB_CANGRE=3;
N_NUM_CANGRE=4;
N_ORI_ARANA=180;
N_PROB_ARANA=3;
N_NUM_ARANA=3;
N_ORI_TABLA=50;
N_PROB_TABLA=3;
N_NUM_TABLA=4;

N_AZAR_ENER=9996;
N_AZAR_DLAT=9996;
N_AZAR_DREC=9996;
N_AZAR_ESCU=9985;


N_VIDA_CHTAB=075;   /// VIDA QUE QUITA
SI DA EN..
N_VIDA_CHARA=070;
N_VIDA_CHBUR=050;
N_VIDA_CHGRAN=500;
N_VIDA_PARED=070;
N_VIDA_DIS3=40;
```

```
N_TOTAL_DLAT=1000;    /// NIVELES
N_TOTAL_DREC=1000;
N_TOTAL_VIDA=1000;
N_IND_DLAT=1000;
N_IND_DREC=1000;
N_IND_VIDA=1000;
N_IND_ESCUS=1;
N_IND_ESCU=0;
           //// DISPAROS ENEMIGOS
N_FREC_TBUR=20;
N_VEL_DBUR=20;
N_VEL_DARA=25;
N_VEL_DHAB=20;

N_FREC_ESCU=25;


End

case 3;
 SD=6;
 ST=4;

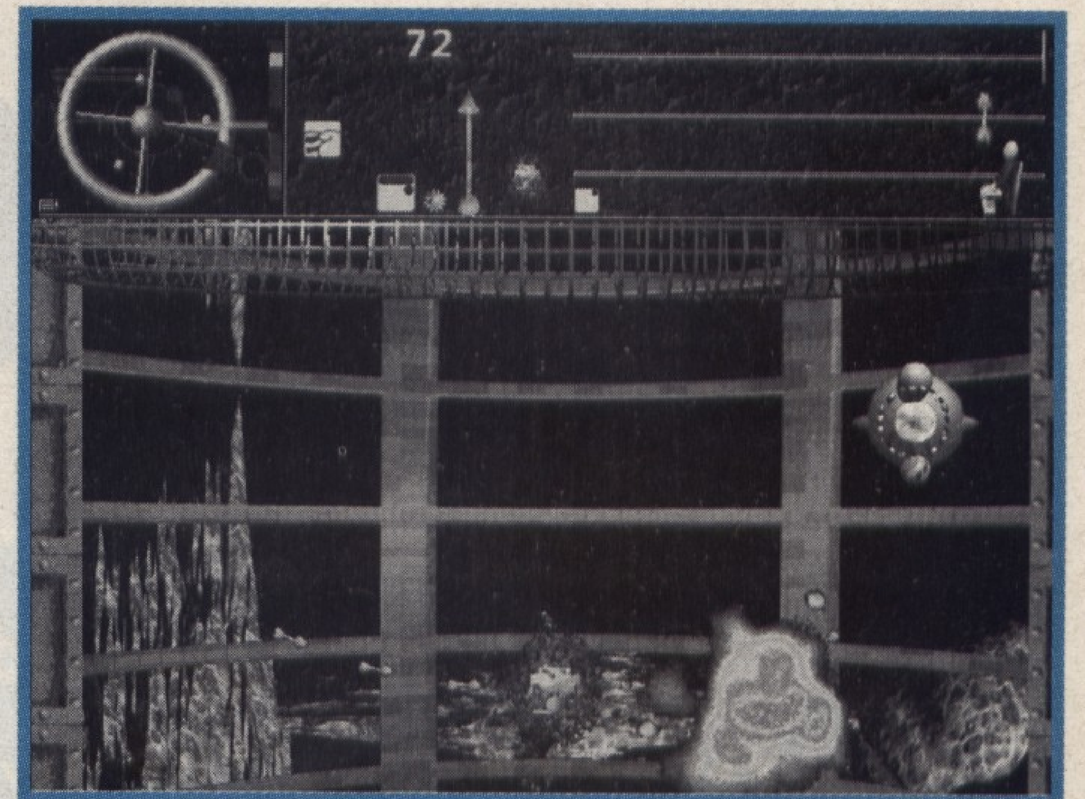
// INDICADORES DE APARICION DE
ENEMIGOS

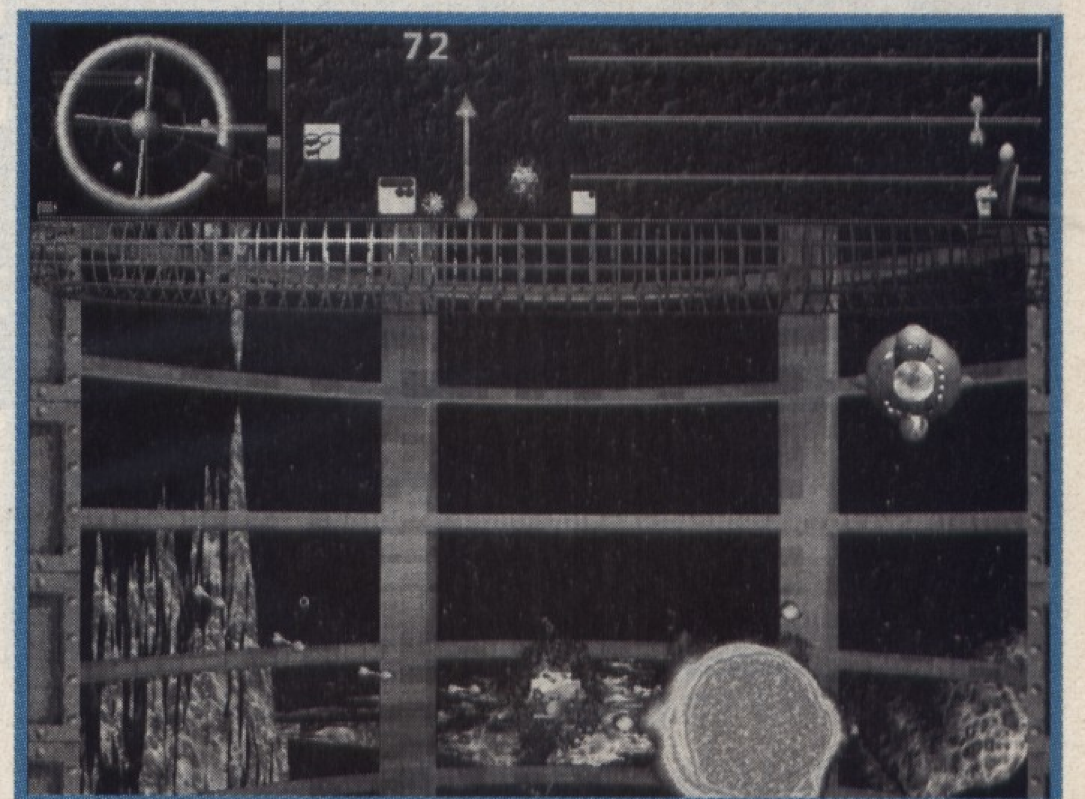
GB1=408;
GB2=437;
GT1=387;
GT2=407;
GA1=180;
GA2=209;
sound(son_fase4,256,256);
N_ORI_CANGRE=0;
N_PROB_CANGRE=3;
N_NUM_CANGRE=4;
N_ORI_ARANA=150;
N_PROB_ARANA=3;
N_NUM_ARANA=3;
N_ORI_TABLA=50;
N_PROB_TABLA=3;
N_NUM_TABLA=4;

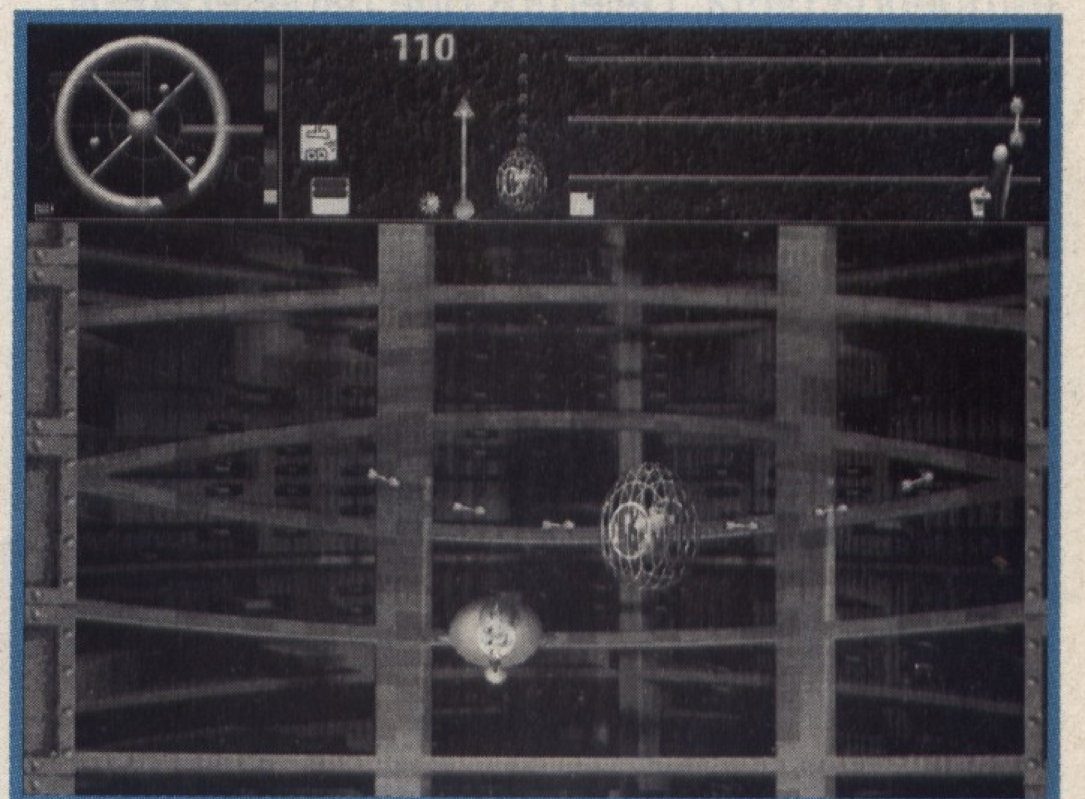
N_AZAR_ENER=9997;
N_AZAR_DLAT=9997;
N_AZAR_DREC=9997;
N_AZAR_ESCU=9990;

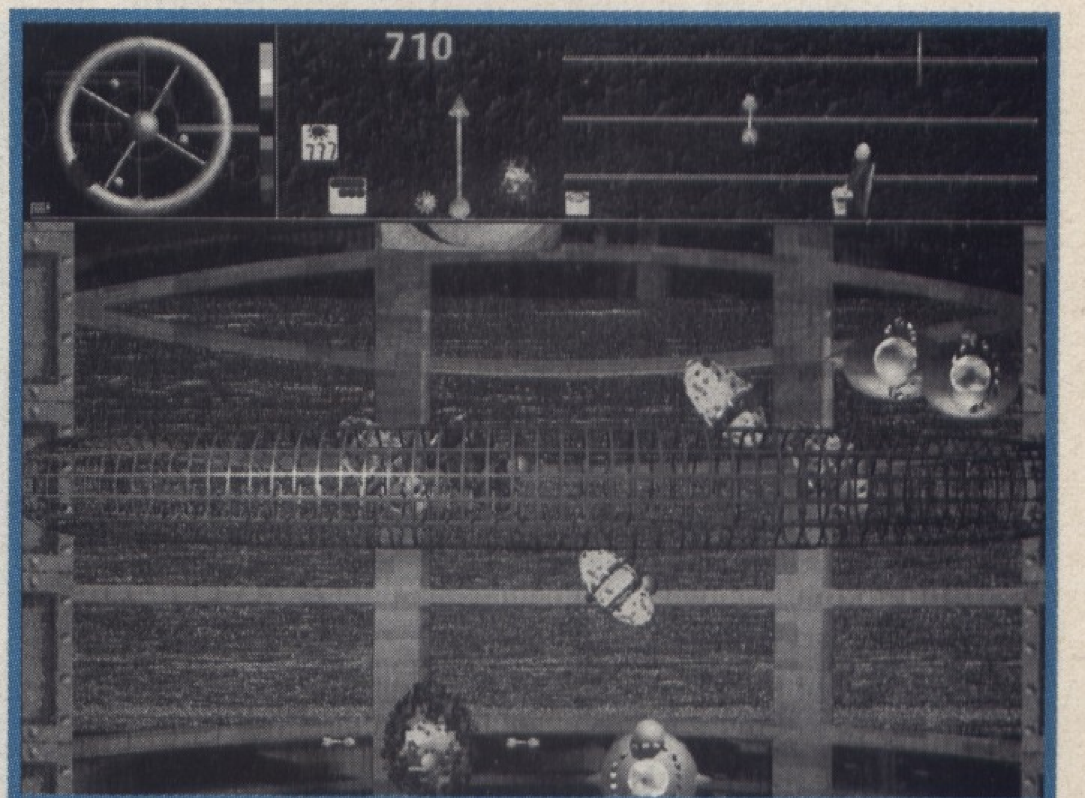

N_VIDA_CHTAB=100;   /// VIDA QUE QUITA
SI DA EN..
N_VIDA_CHARA=100;
N_VIDA_CHBUR=100;
N_VIDA_CHGRAN=0700;
N_VIDA_PARED=100;
N_VIDA_DIS3=050;

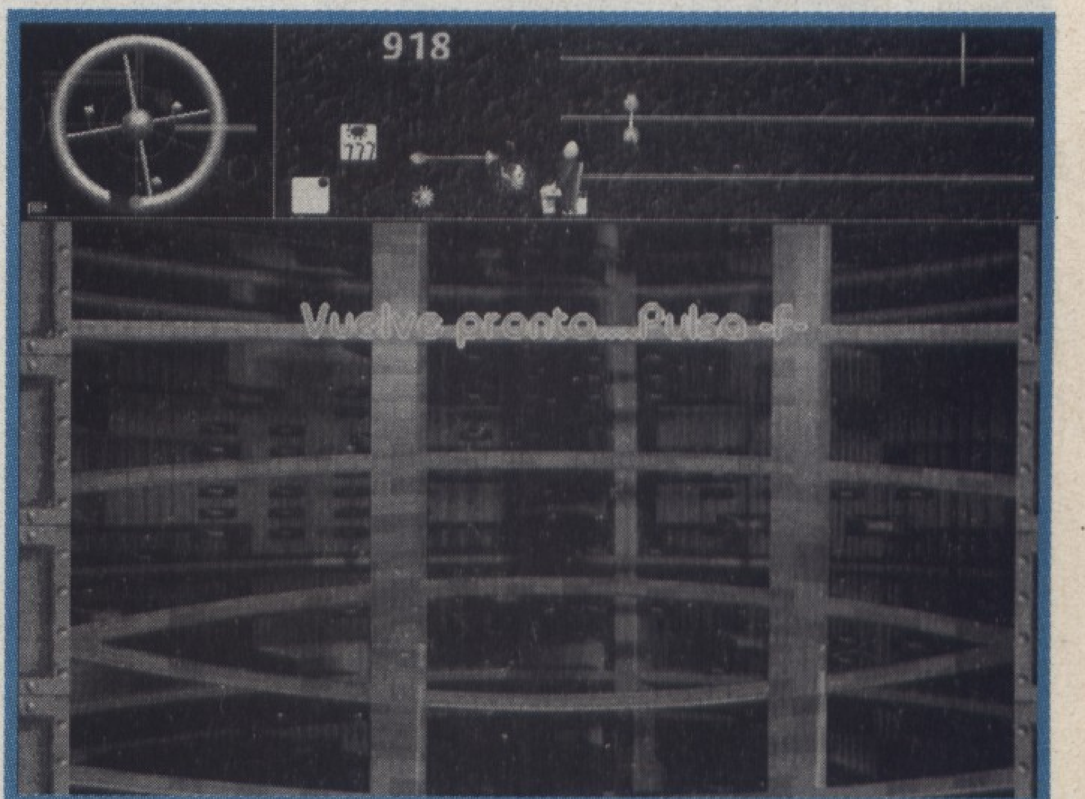
N_TOTAL_DLAT=1000;    /// NIVELES
N_TOTAL_DREC=1000;
N_TOTAL_VIDA=1000;
```











Autor: Emilio José Lopera Joye

Men In Green

El segundo afortunado ha sido un juego tipo *Quake*. Sólo es una versión beta del juego, pero lo que hemos visto nos parece de la suficiente calidad como para merecer este segundo premio. Seguro que con un poco más de trabajo, *Men In Green*, será un estupendo juego, de esos en lo único que tenemos que pensar es a quién disparar primero.

Con lo fácil que es programar arcades 3D con DIV y DIV 2 nos extraña que hasta ahora nadie nos haya enviado una creación propia basada en este género. Pero parece que ya nos ha llegado el primer trabajo y tiene una estupenda pinta. Dejemos que hable el autor:
"Hola gente aquí los "Men In Green" para presentaros este juego, bueno, es la demo, el primer nivel al completo y el mapeado del segundo (tened cuidado porque en algunos sitios no podéis salir).

Con Men In Green podrás jugar en primera o en tercera persona

"La historia trata de una organización tan secreta que ni siquiera los que pertenecen a ella saben quienes son; pero ahora los "TIPOS MALOS" también llamados "BAD BOYS" están fastidiando un poco a la gente de la organización y hay que evitarlo.
"En esta versión solamente está habilitado el modo de 1 jugador, en el que se puede elegir entre los personajes que interpreta EJ y los que hace AM, la misión consiste en conseguir una tarjeta de teletransporte, que se ha de usar al lado de un "tarugo" situado en una sala pequeña que está tras abrir la ultima puerta, una de ellas has de abrirla hablando con los policías.
"También hay que conseguir las 25 chapas de MIG para que aparezca en otro lugar del





decorado "LA CHAPA", una chapa dorada q haría falta para las fases de bonus, pero com no hay..., pero de todas maneras buscadlas todas... ya que no son muy fáciles de encontrar.
"Los controles del juego son:

- Las flechas direccionales para mover al personaje (no intentes andar hacia atrás).
- El boton 1 del mando o la tecla "j" para disparar.
- El boton 2 del mando o la tecla "k" junto con: "arriba" para saltar hacia delante, "abajo" para dar volteretas hacia atrás.
- El boton 3 del mando o la tecla "i" para activar los ascensores o usar objetos (la tarjeta para teletransporte).
- El boton 4 o la tecla "h" que se usa junto: "arriba" para andar sigilosamente; "dispar para, en vista de 1ª persona, usar las gafas visión especial, para que con las teclas direccionales veamos lo que hay alrededor. Para quitarlas, pulsa "salto".
- Con la fecla "F9" hace que veáis todo en 1ª persona y la tecla "F10" en 3ª; cuando queráis podéis probar esas MIGfases que hemos preparado.

"Si quereis contactar estamos en el escondite secreto del garito 21, en la cueva de abajo (ups!!!, se me ha escapado; ahora nos iremos la taberna de la esquina al ático, ups!!! otra vez)".
Y aquí está un jugoso pedazo del código fuente que seguro que os ayuda si estáis programando algún juego de este tipo.





apa dorada que
us, pero como
s buscadlas
es de

mover al
acia atrás).
"j" para

"k" junto
delante,
a atrás.
"i" para
etos (la

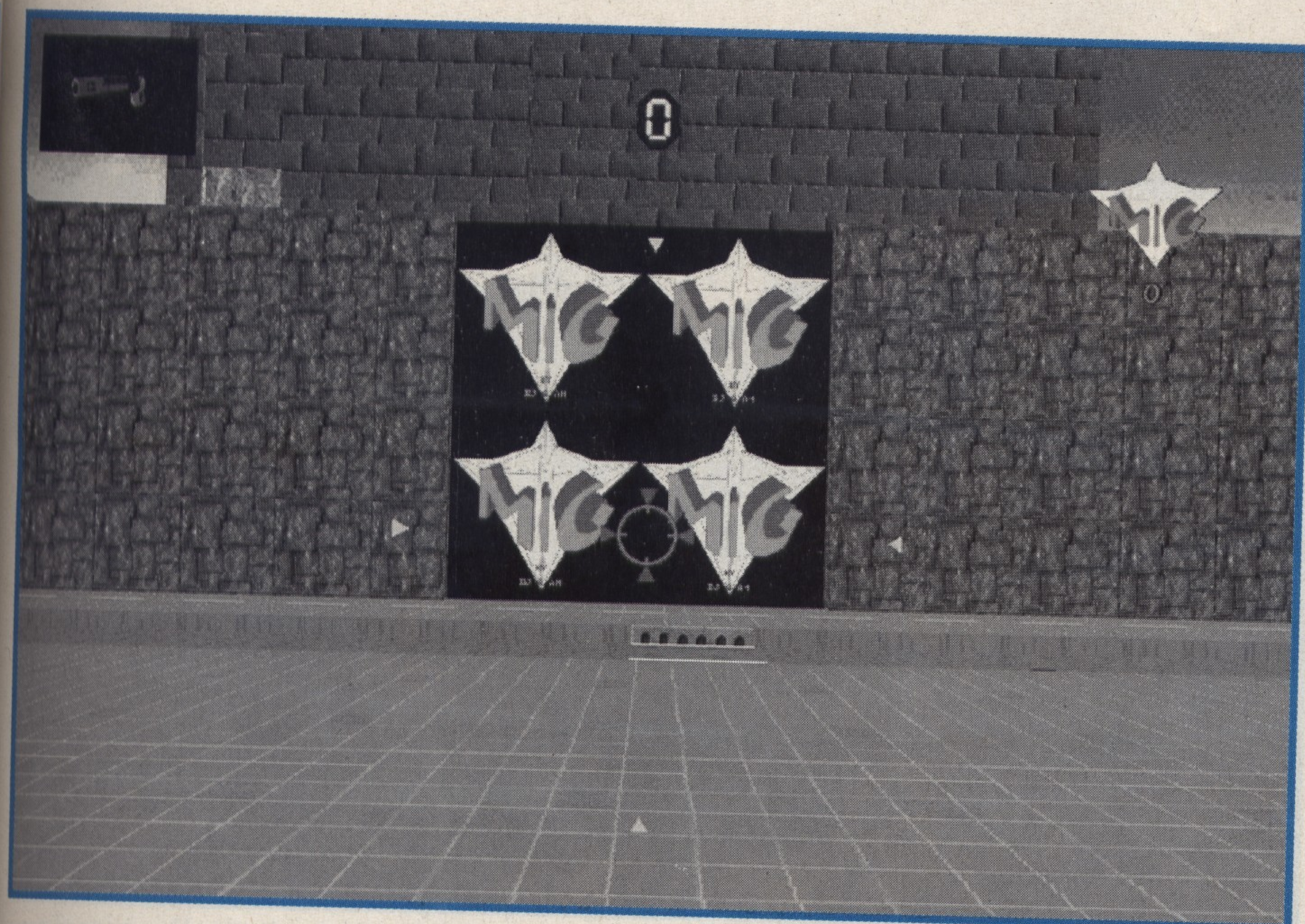
usa junto:
e; "disparo"
r las gafas de
teclas
alrededor.

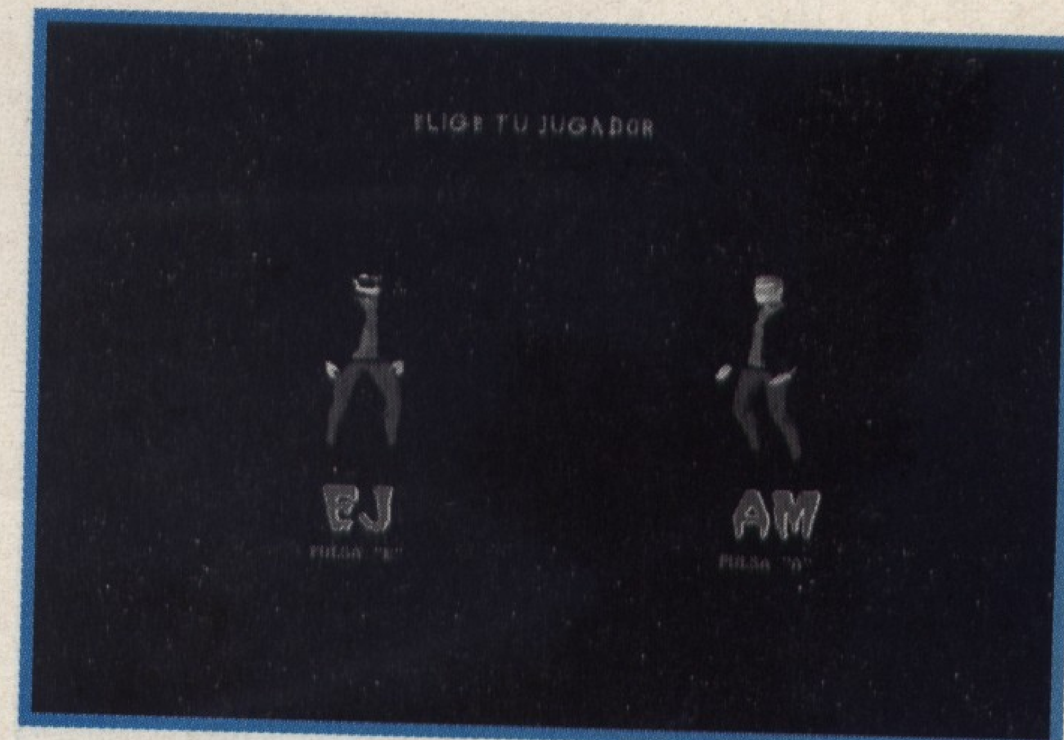
todo en 1ª
ando
ses que

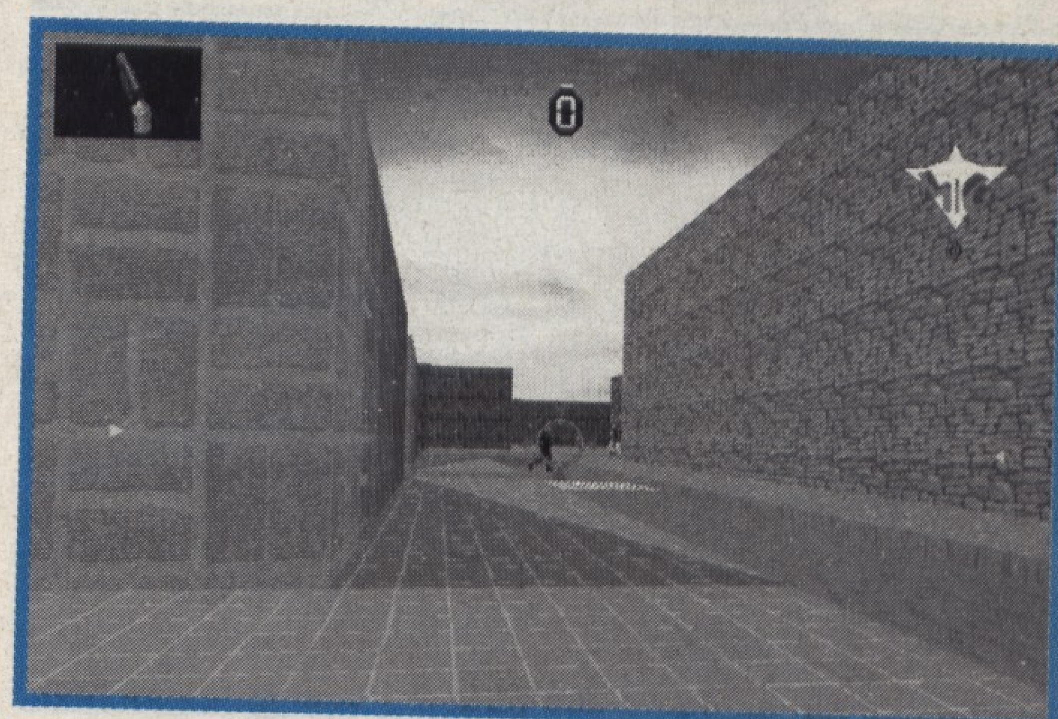
escondite
e abajo
os iremos a
!!! otra

digo
stáis
bo.









```
PROGRAM MIgv15;

  GLOBAL
   HELI_CHANEL;
   estado_fase;
   fondito;

   viejas_chapas;
   viejo_stage;
   viejos_puntos;
   viejo_personaje;

   chapas1;
   chapas2;
   chapas3;
   chapas4;
   chapas5;
   CHAPAS6;
   chapas7;
   chapas8;

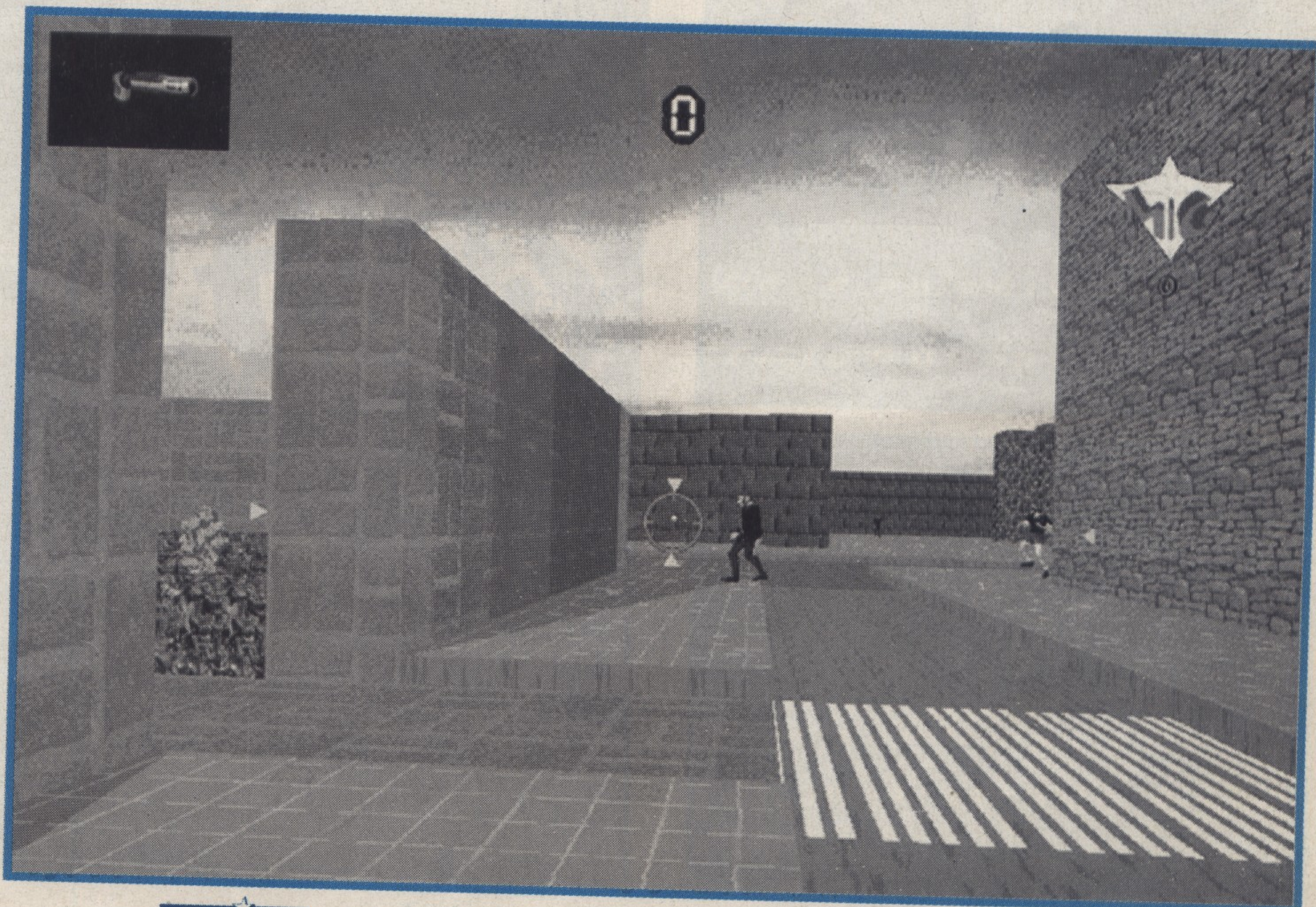
   STAGE1;
   STAGE2;
   STAGE3;
   STAGE4;
   STAGE5;
   STAGE6;
   stage7;
   stage8;

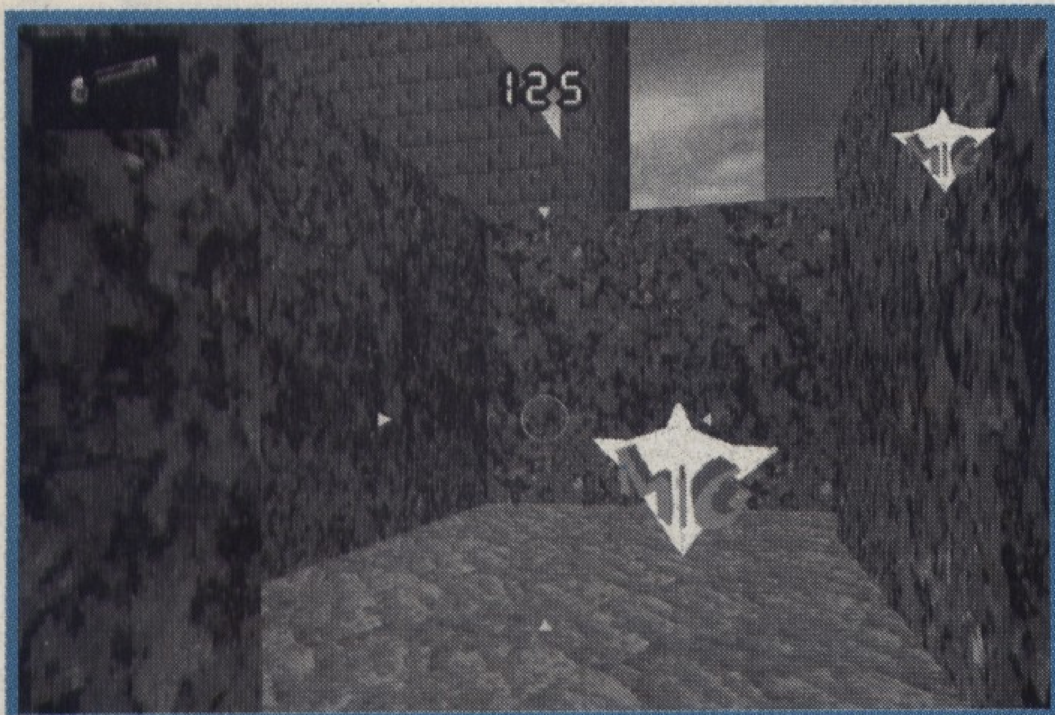
   puntos1;
   puntos2;
      puntos3;
   puntos4;
   puntos5;
   puntos6;
   puntos7;
   puntos8;

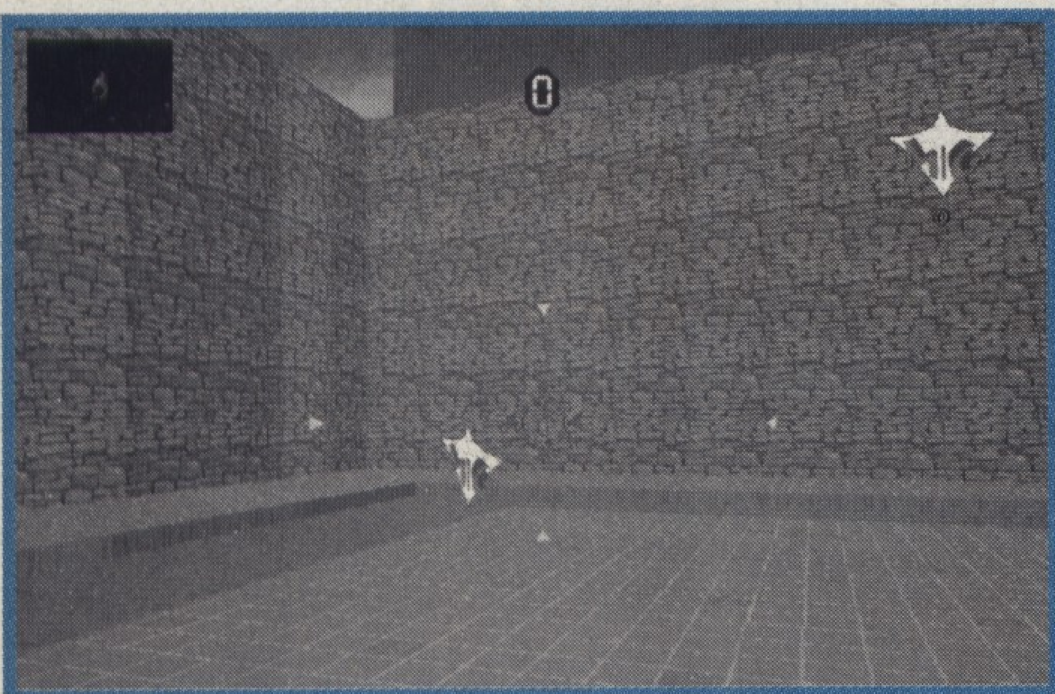
   PERSONAJE1;
   PERSONAJE2;
   PERSONAJE3;
   PERSONAJE4;
   PERSONAJE5;
   PERSONAJE6;
   personaje7;
   personaje8;

   z_camarax;
   ALEATORIO1;
   ONDAF;
   Z_ONDA;
   TIPO_ARMA;
   z_bola;
   SIGUIENTE_SECTOR;

   puerta;
   MALO1;
   bola;
   angulo_a_ej;
   ESTADO_POLICIA2;
   pared_ej;
   textura;
   posiciones;
   CHAPAS_DORADAS;
   CHAPAS_EJ;
   PUNTOS_EJ;
   BANDERA;
   STAGE;
   vida_ej;
   player;
   x_am;
```








```
y_am;
z_am;
EMI;
ESTADO_EJ;
ESTADO_MENU;
FASE_JUEGO;

camaravirgue;
estado_camara1;
estado_camara2;
POSICION_CAMARA1;
POSICION_CAMARA2;
camara_externa1;
angulo_ej;
x_ej;
y_ej;
z_ej;
TIPO_FASE;
angulo_camara1;
sector_camara1;
sector_ej;
suelo1;
suelo2;
techo1;
techo2;
diferencia;
suma_x;
suma_y;

BEGIN

SET_MODE(M800X600);
INTRO();

FRAME;
END
```

```
//————————————————————
// PROCESO DE LA INTRO
//————————————————————

PROCESS INTRO();

BEGIN

SET_FPS(200,0);
load_wav("mig/intromig.wav",0);
LOAD_FPG("MIG_INTR.FPG");

DE1();

REPEAT
FRAME;

UNTIL(KEY(_ENTER));
 PRINCIPIO();

FRAME;
END

//———————————
// proceso de1();
//———————————‡

process de1();
 begin
  file=0;
  set_fps(100,0);
 x=400;
 y=-60;
  posiciones=0;
```

```
loop
 graph=15;
 y+=10;

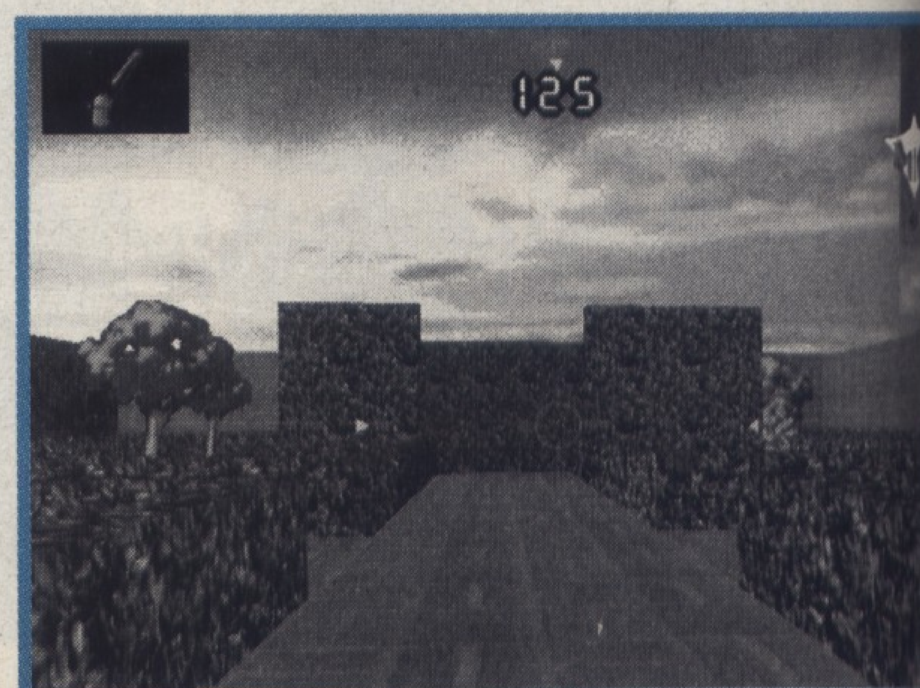
 posiciones+=10;
if (posiciones==300);
 sound(0,256,256);
 los();
 frame;
 end
 frame;
 end
 frame;
 end

process los();

 private
 position;

begin

graph=16;
position=0;
```



Autor: Félix Martínez Vera

Numbers

Un curioso juego de plataformas donde la originalidad y la creatividad de su autor brillan a gran altura. Los protagonistas de este título no es ningún muñeco animado, son los números, que deberán ir salvando todos los obstáculos que aparezcan y matando toda alimaña que intente impedir su progreso.

El juego tiene seis fases, cada una con un jefe final, y un jefe final de los finales. Las teclas son los cursores para la dirección, y el espacio para la acción. El salto es el cursor arriba. El cursor abajo sólo lo utiliza el número cero cuando está deslizándose y quieres que se detenga.

Cada personaje (número) tiene una habilidad concreta y en determinados lugares de cada pantalla sólo uno o dos números serán capaces de pasar por ese sitio.

Los bonus (números que caen del cielo) amarillos cambian de personaje sumando ese número que ha caído al número que tu llevas. Si pasa de nueve no hace nada. Por ejemplo si llevas el tres y coges el número amarillo cuatro pasaras a llevar el número siete.

Los bonus rojos hacen lo mismo pero restando.

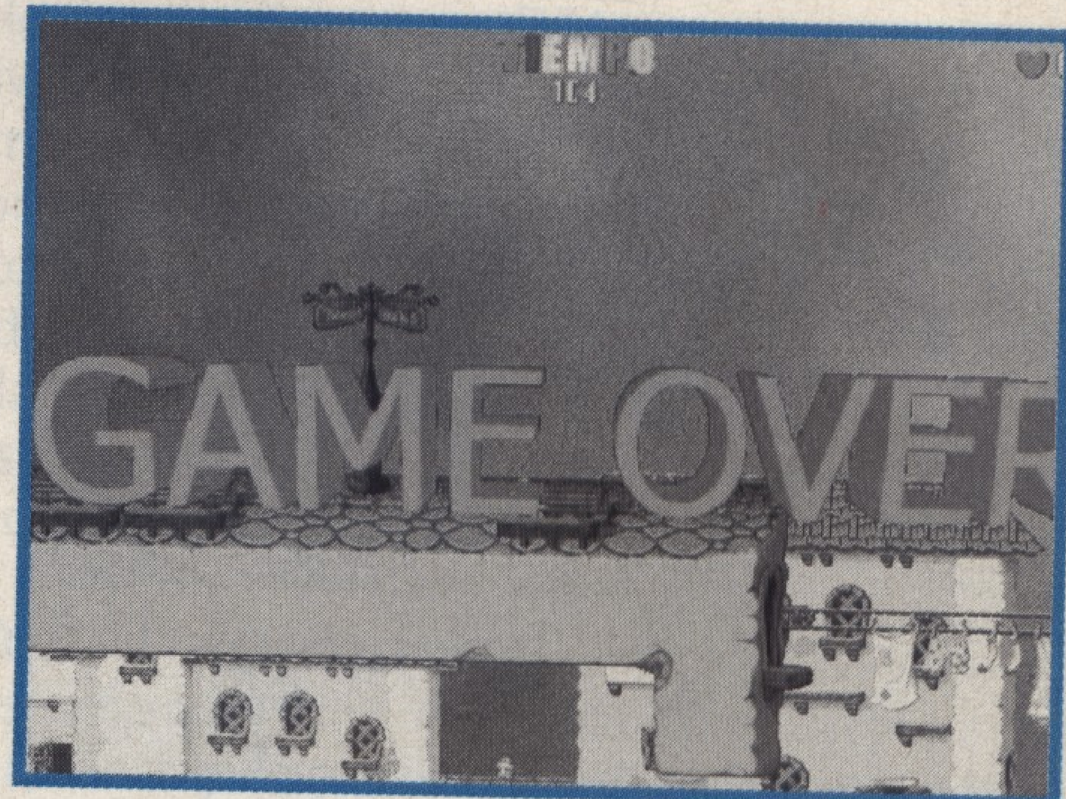
El bonus verde (la interrogación) te convierte en un número al azar (para situaciones desesperadas).

El bonus azul es el de reserva, es decir que si coges el bonus azul 8, cuando pulses la tecla 8 te convertiras en el personaje número 8.

En numbers los protagonistas son guarismos matemáticos, nada de Laras o Marios

Los jefes finales de fase sólo se les destruye por medio de disparos, no intentes nada con otros números porque perderás una vida.

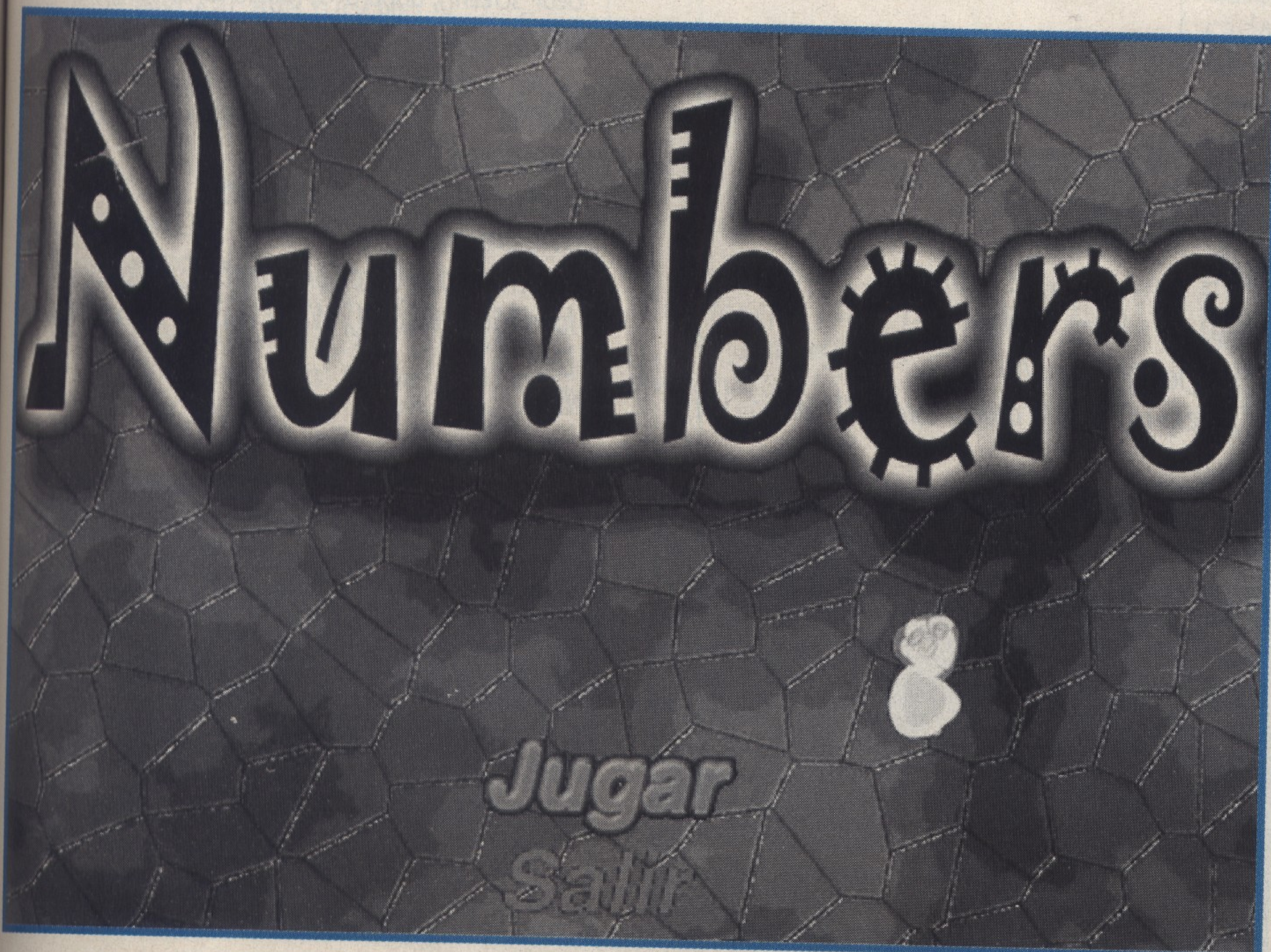
Aquí os ponemos una parte del código fuente de este juego que podéis encontrar íntegro en el CD-Rom:

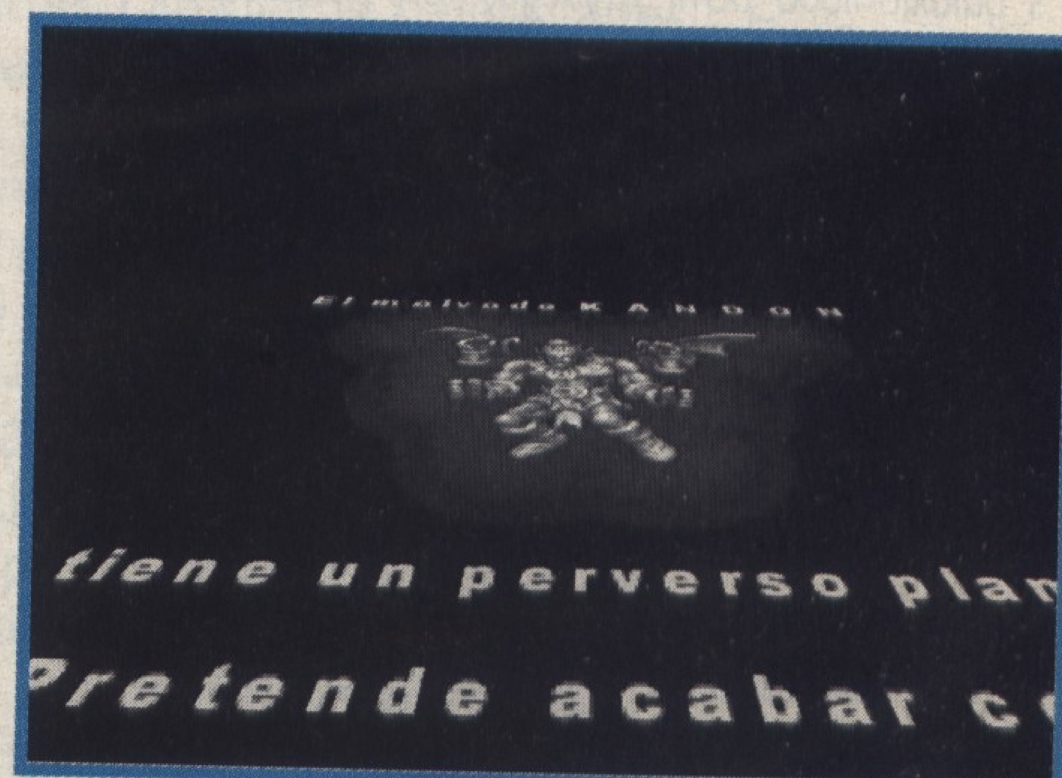


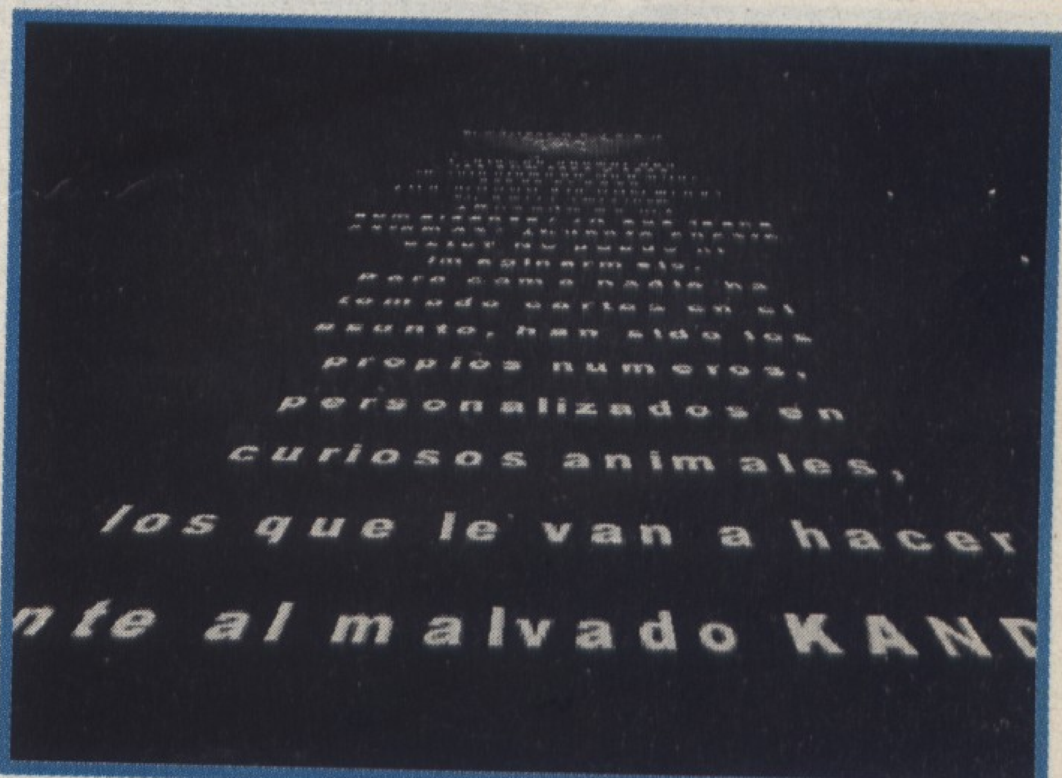

```
// Juego creado integramente con DIV 1.03b

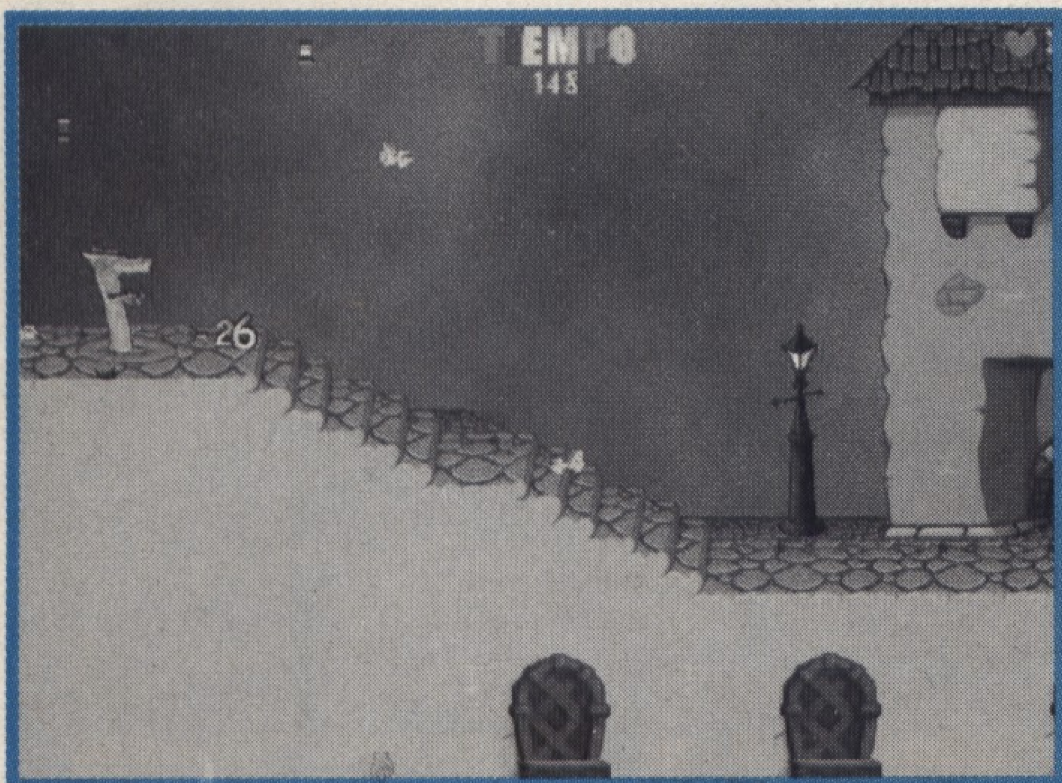
PROGRAM Numbers;

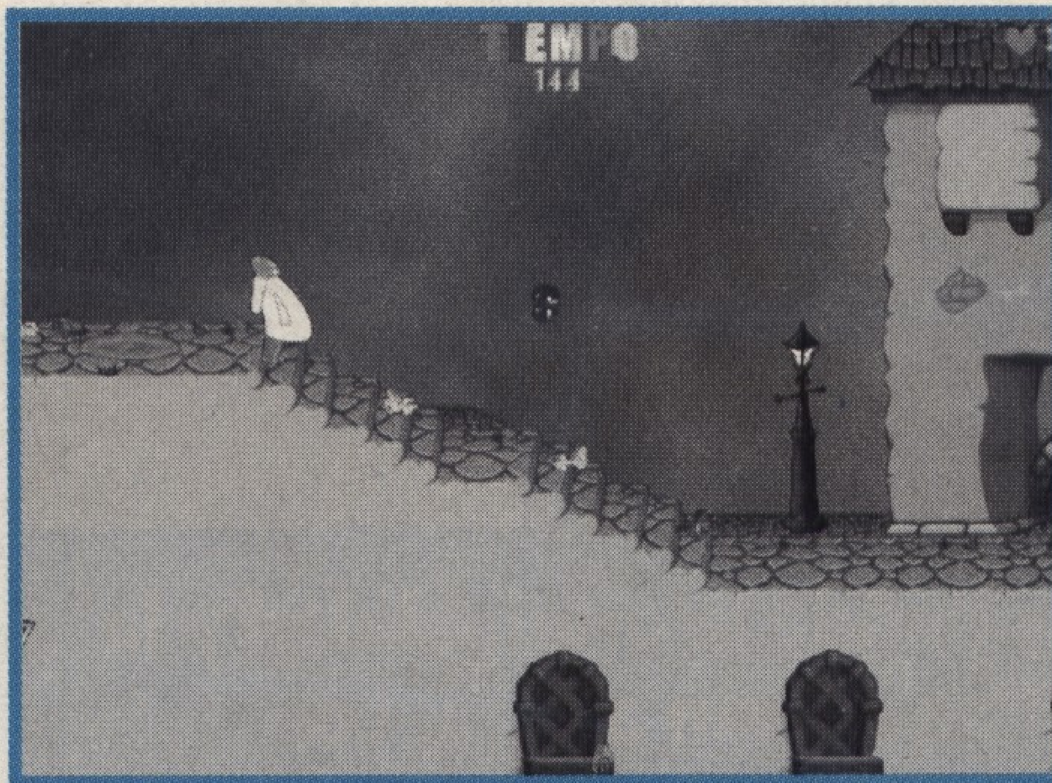
GLOBAL
idprota; idenemigo; fase; altura=42; ar; vuelta; idedo;
numero_prota;
cambiar_prota=1; diferencia; ener_boss;
numero_bonus=0; cuatro_salto;
xfinal; vidas; disparo=1; coordenadas; fiche_duro; fantas;
fichero;
fichero1; fichero_protas; sonando; fichero_general;
fuente1; musica; m_fondo;
tiempo; musica_intro; s_salto; s_explosion; s_calavera;
s_galaxia; s_pisoton;
s_jefe4to; s_uno; s_cinco; s_siete; s_nueve; s_chapuz;
s_arder; s_hurry;
s_grito; s_gameover; s_tigre; s_pause; s_rata; s_ala;
s_alas; s_exploss;
s_pinchos; s_rojo; s_verde; s_amarillo; s_azul; s_vida;
s_tiempo; s_nada;
s_dedo; s_dragon1; s_dragon2, s_dragon3; s_lanzar;
s_trueno;s_jefedos;
s_salto1; s_caida1; s_saliagua;
```

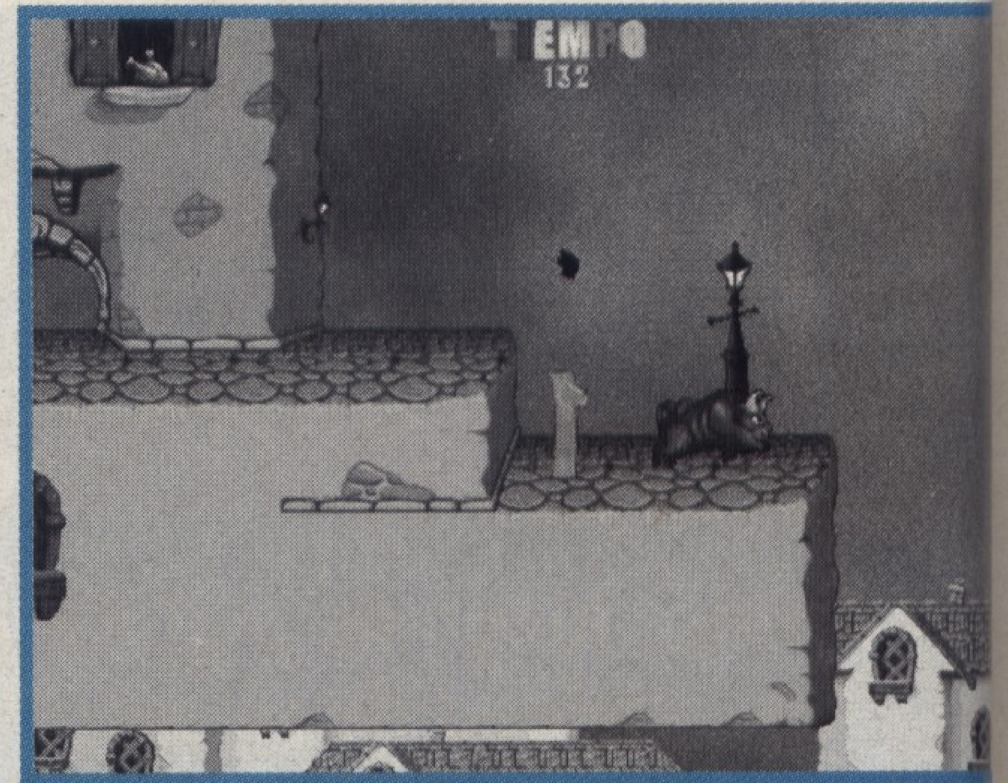












```
rugido; seis_roca; dos_alto; oxigeno=400; en_agua=0;
barra_activa=0; oxigeno_temp; diez;
coreo1[3]=3,0,1,2;
coreo2[4]=4,0,1,2,1;

LOCAL
pri_mov; ult_mov; fuerza_salto;
incx; mortal;
velocidad_gravedad=0;
en_suelo=FALSE;

BEGIN
  set_mode(m640x480);

  fichero_general=load_fpg("c:\juegos\vera\platafor.fpg");
  fichero1=load_fpg("c:\juegos\vera\numeros.fpg"); //
Gr ficos de los bonus
  fichero_protas=load_fpg("c:\juegos\vera\protas.fpg");
// Gr ficos de los protas

  musica_intro=load_pcm("c:\juegos\vera\intro.pcm",1);

  s_calavera=load_pcm("c:\juegos\vera\primera.pcm",0);
  s_arder=load_pcm("c:\juegos\vera\fuego.pcm",0);
  s_galaxia=load_pcm("c:\juegos\vera\galaxia.pcm",0);
  s_salto=load_pcm("c:\juegos\vera\salto.pcm",0);
  s_explosion=load_pcm("c:\juegos\vera\explosio.pcm",0);
  s_chapuz=load_pcm("c:\juegos\vera\chapuzon.pcm",0);
  s_hurry=load_pcm("c:\juegos\vera\harryup.pcm",0);
  s_uno=load_pcm("c:\juegos\vera\rugido.pcm",0);
  s_cinco=load_pcm("c:\juegos\vera\buceo.pcm",0);
  s_siete=load_pcm("c:\juegos\vera\disparo7.pcm",0);
  s_nueve=load_pcm("c:\juegos\vera\rugido9.pcm",0);
  s_rojo=load_pcm("c:\juegos\vera\rojo.pcm",0);
  s_amarillo=load_pcm("c:\juegos\vera\amarillo.pcm",0);
  s_azul=load_pcm("c:\juegos\vera\azul.pcm",0);
  s_tiempo=load_pcm("c:\juegos\vera\tiempo.pcm",0);
  s_verde=load_pcm("c:\juegos\vera\verde.pcm",0);
  s_vida=load_pcm("c:\juegos\vera\vida.pcm",0);
  s_nada=load_pcm("c:\juegos\vera\nada.pcm",0);
  s_dedo=load_pcm("c:\juegos\vera\dedo.pcm",0);
  s_grito=load_pcm("c:\juegos\vera\grito.pcm",0);
  s_pinchos=load_pcm("c:\juegos\vera\pinchos.pcm",0);
  s_gameover=load_pcm("c:\juegos\vera\gameover.pcm
",0);
  s_pause=load_pcm("c:\juegos\vera\pause.pcm",0);
  s_ala=load_pcm("c:\juegos\vera\ala.pcm",0);
  s_exploss=load_pcm("c:\juegos\vera\exploss.pcm",0);

  fuente1=load_fnt("c:\juegos\vera\plata1.fnt");

  video();
END

/*
ÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄ
ÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄ
                    Control general del juego
*/

PROCESS control();


PRIVATE
id_txt; contar; soti;
BEGIN

  fase++;
  Pinta_fase(); fantas=0;
 IF (fase<>14) teclas(); dedo(); corazon(); tiempo=151;
tiempo_gra();
  Write_int(1,610,0,0,&vidas);
  LOOP

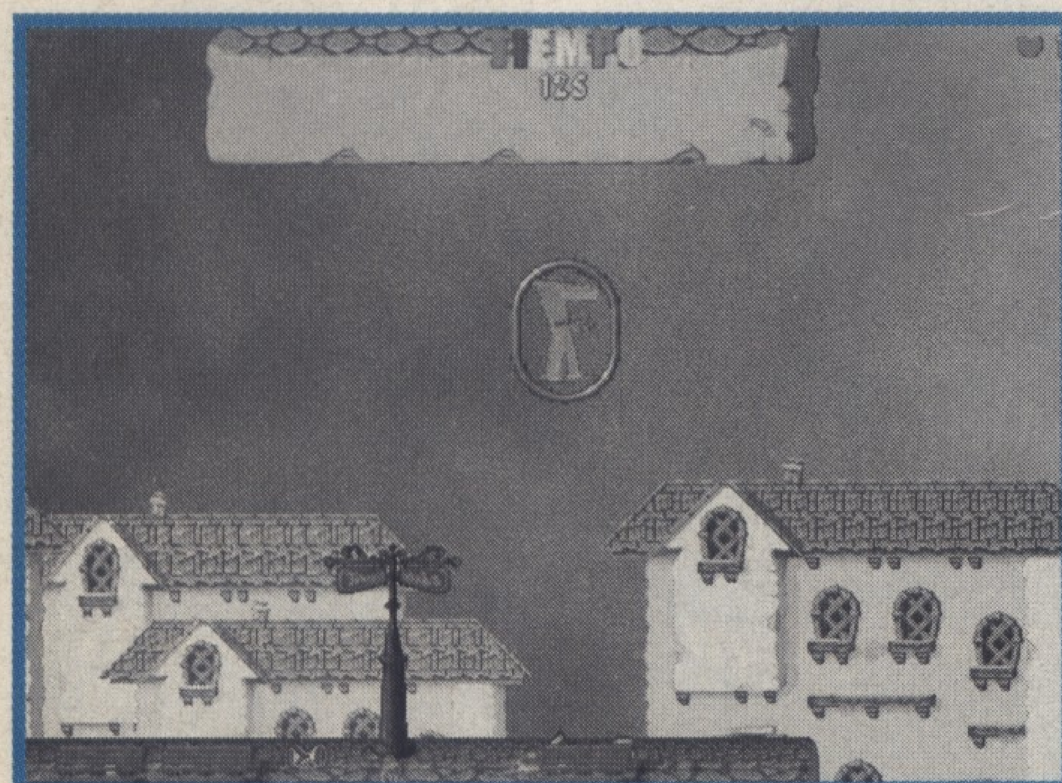
  IF (timer[0]>100) timer[0]=0; tiempo—; END
  IF (key(_esc)) escape(); END

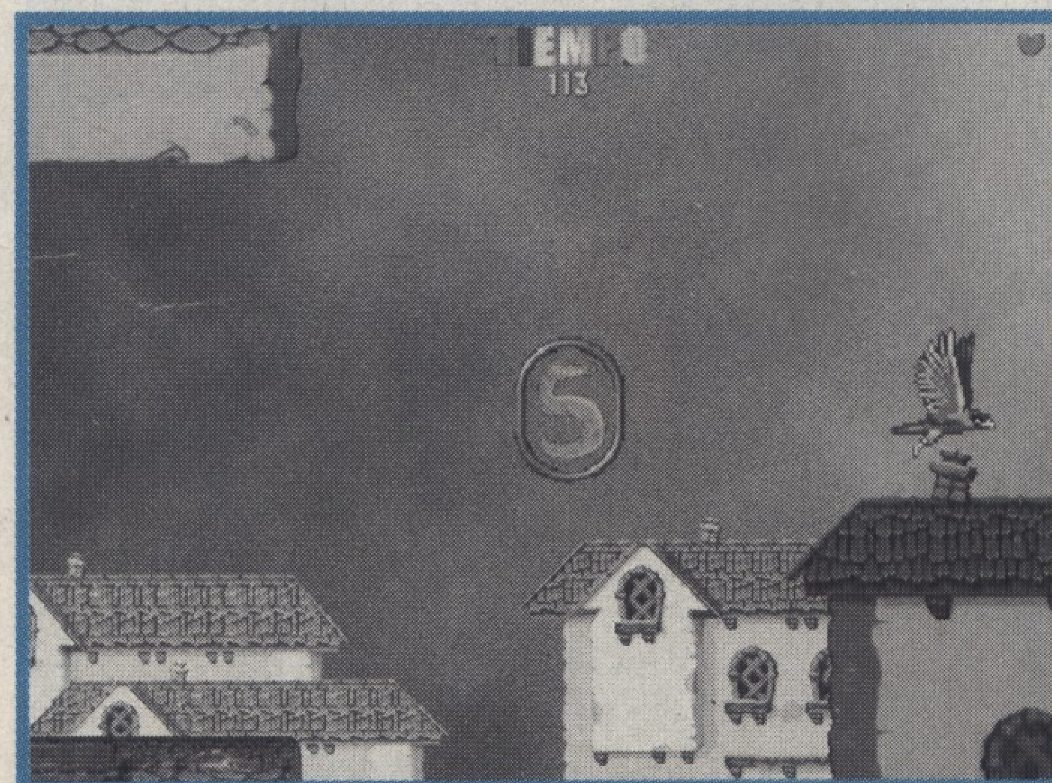
  IF (tiempo<=20) IF (soti++==30)
sound(s_hurry,156,256); soti=0; END END
  IF (tiempo<=0) game_over(); end

  IF (rand(1,5000)>3850)
   FROM contar=1 to 9;
    IF (timer[contar]>30 and numero_bonus<9)
numero_bonus++;
      bonus (rand(idprota.x-320,idprota.x+320),-
40,rand(1,31),contar); break;
     END
    END
   END

  IF (idprota.x>=xfinal) vuelta=0; let_me_alone();
  IF (fase!=13) fade(0,0,0,6); WHILE (fading) frame; END
END
  clear_screen(); disparo=1; rugido=0; cargar();
  END

  FRAME;
  END // loop
  delete_text(all_text); stop_scroll(0);
stop_sound(musica);
  numero_bonus=0; cambiar_prota=1; opciones();
  fade(100,100,100,4); WHILE (fading) frame; END
  END
END
```









Efecto de agua sobre una imagen.

que se requieran antes de finalizar la ejecución del salvapantallas. Si no realizamos ninguna de estas acciones podemos omitir esta función.

- *Ss_time:* esta variable contiene un valor en centésimas de segundo que nos indica el tiempo de espera del salvapantallas. Debe ser definida en la función ss_init().
- *Ss_exit:* esta variable contiene el valor 0 durante el período de ejecución del salvapantallas. Podemos cambiar su valor a 1 durante la ejecución de la función ss_frame() para de esta forma interrumpir la ejecución del salvapantallas. Además se pondrá a 1 su valor cuando el usuario pulse una tecla, se active el ratón o el joystick.
- *Ss_status:* nos indica cuál es el estado actual del salvapantallas. Su valor es 0 si se encuentra desactivado y 1 si está activado. Esta variable la podemos utilizar en otro tipo de DLLs para ver si se encuentra activo el salvapantallas.

Para utilizar cualquiera de las funciones anteriormente descritas, debemos indicarle a DIV que están disponibles y las hemos definido. Para ello se utiliza la ya conocida función *DIV_export()*.

Además de esta función, debemos notar que se tratan de DLLs de autocarga, con lo que debemos indicárselo a DIV. Para ello debemos invocar a la función Autoload() dentro del bloque divmain. Veamos cómo sería el código necesario para ello:

```
Void __export divmain
(COMMON_PARAMS) {
    AutoLoad();
    GLOBAL_IMPORT();
    DIV_export("ss_init",ss_init);
    DIV_export("ss_frame",ss_frame);
    DIV_export("ss_end",ss_end);
}
```

Algunos puntos a tener en cuenta

A la hora de crear un salvapantallas debemos tener en cuenta varios aspectos:

- Podemos alterar la pantalla a nuestro antojo y no nos debemos preocupar por el contenido anterior a la ejecución del salvapantallas, ya que DIV se encarga automáticamente de restaurar el contenido anterior.
- El tiempo de espera podemos especificarlo colocando un valor en la variable *ss_time*, pero si no, su valor por defecto será de 30 segundos *(ss_time = 3000;)*.
- Las únicas funciones que debemos especificar forzosamente son *ss_init* y *ss_frame*.

Un ejemplo sencillo

Vamos a ver en primer lugar un ejemplo sencillo que nos permitirá hacer nuestro pequeño primer salvapantallas. Éste consistirá en un efecto de agua como el generado por un televisor cuando se encuentra desintonizado. Para ello, debemos colocar cada punto de la pantalla al azar, calculándolos en cada nuevo frame. No importa demasiado la paleta que usemos, aunque una escala de grises sería muy recomendable. No obstante, y por razones de sencillez, nuestro ejemplo obviará el uso de una paleta apropiada. Para hacer todo esto aplicaremos los conceptos aprendidos en los números anteriores sobre el uso del puntero al buffer de la pantalla *buffer[]* y las constantes *wide* y *height* para conocer las dimensiones de la pantalla. Veamos el código que usaríamos:

```
#include <stdio.h>

#define GLOBALS
#include "div.h"

void ss_init(void) {}

void ss_frame(void)
{
int i,j;

for (int i = 0; i < width ; i ++)
   for (int j = 0; j < height ; j++)
      buffer[j*width+i] =
div_rand(0,255);
}

void __export divmain
(COMMON_PARAMS) {
   AutoLoad();
   GLOBAL_IMPORT();
   DIV_export("ss_init",ss_init);
   DIV_export("ss_frame",ss_frame);
}
```

¿Fácil verdad? Como se puede apreciar, la función de inicialización no contiene ningún código. Podríamos incluir en esta parte el tiempo de activación o de espera del salvapantallas incluyendo la sentencia:

ss_time = 1500;

con lo que le diríamos a DIV que activase nuestro salvapantallas transcurridos 15 segundos de espera. Si por ejemplo queremos crear un efecto de gradiente o gama de color y deseamos tener una paleta ordenada, podemos hacer las ordenaciones previas necesarias en la función de inicialización.

Para más información

Podemos encontrar otro ejemplo más interesante y complejo de salvapantallas en la colección de ejemplo que acompaña al CD original de DIV. Se trata de SS1.DLL, que realiza un complejo efecto de granulado que no comentaremos en este número por lo extenso de su código. Veamos ahora cómo podemos programar el otro tipo de DLL, no tratado hasta ahora, como son las librerías de autocarga, semejantes a los salvapantallas en cuanto a programación.

DLLs de autocarga

Tal vez un tema no más interesante pero sí más útil son las librerías de autocarga. Cargándose de forma automática, éstas nos permiten tener un efecto en pantalla de forma permanente tratando lo que se denominan puntos de ruptura o *entry-points*. Mediante estos podemos interferir de forma directa en cualquier programa en el proceso que deseemos: realizar alteraciones en cada frame, cambiar la paleta, alterar mapas... Muchos de ellos nos sonarán por haberlos utilizado anteriormente en la creación de DLLs de funciones. Así que veamos cómo se programan.

A la hora de crear un salvapantallas debemos tener en cuenta varias cosas

Programando las DLLs de autocarga

El concepto es muy parecido al de la programación de salvapantallas, sólo que en lugar de exportar las funciones *ss_init()* y *ss_frame()*, exportamos los puntos de entrada, que junto con sus efectos podemos verlo en la tabla 1. La forma de indicar que vamos a usar un punto de ruptura se hace mediante la exportación usual de funciones: *DIV_export()*.

Además de esto y al igual que en los salvapantallas, debemos invocar en la función *divmain*, a la función *AutoLoad()*, que se encarga de indicar a DIV que se trata de una DLL de autocarga.

Tabla 1. Relación de entry-points en DIV.

Función de entry-point	Efecto
Set_video_mode()	Cambia a un nuevo modo de pantalla
process_palette()	Modifica una paleta
process_active_palette()	Modifica la paleta activa
*process_sound(char *sound,int lenght)*	Modifica un nuevo efecto de sonido
*process_fpg(char *fpg,int fpg_lenght);*	Modifica un nuevo fichero
*process_map(char *map,int map_lenght);*	Modifica un nuevo mapa
*process_fnt(char *fnt,int fnt_lenght);*	Modifica una nueva fuente de texto
background_to_buffer(void);	Volcado del fondo al buffer de video
buffer_to_video(void);	Volcado del buffer de video a la pantalla
post_process_scroll(void);	Aplica un efecto a una ventana de scroll
post_process_m7(void);	Aplica un efecto a una ventana de modo7
post_process_buffer(void);	Aplica un efecto al buffer de video
post_process(void);	Modifica las variables de un proceso
*put_sprite(byte * si, int x, int y, int an, int al, int xg, int yg, int ang, int size, int flags);*	Pone un sprite en pantalla.

Un ejemplo de Autocarga

Como hemos comprobado, no hay gran diferencia entre estas librerías y los salvapantallas. De hecho, podemos considerar a los salvapantallas como un tipo específico de librería de autocarga. Vamos a ver un ejemplo sencillo, como es el efecto de agua que acompaña al CD original de DIV. Lo podemos encontrar en el archivo agua.DLL del directorio DLL. Este efecto consiste en, tomada una zona de la pantalla, estirarla o encogerla de forma que parezca que se está balanceando como si de la superficie del mar se tratara, reflejando la imagen que se encuentra justo por encima de la superficie. ¿Cómo hacemos esto? Pues básicamente tomando la zona inferior de la pantalla como zona donde aplicar el efecto de agua. Cada línea de la pantalla se procesa de forma que según un factor de elasticidad aparecerá una o ninguna vez, de forma que se de esa sensación de compresión y descompresión de la imagen. Veamos el código fuente para hacernos una idea aproximada:

```
#include <math.h>
#include <stdio.h>
#include <mem.h>

#define GLOBALS
#include "div.h"

#define ALTURA 20

char    *agua=NULL;

void post_process_buffer(void) {
  static int desp=3000;
  static int dec=-1;
  static int valor=0;
  int    x,inc;
  FILE   *f;

  if(valor==0) {
    valor=1;
    agua=(char
*)div_malloc(ALTURA*wide);
  }

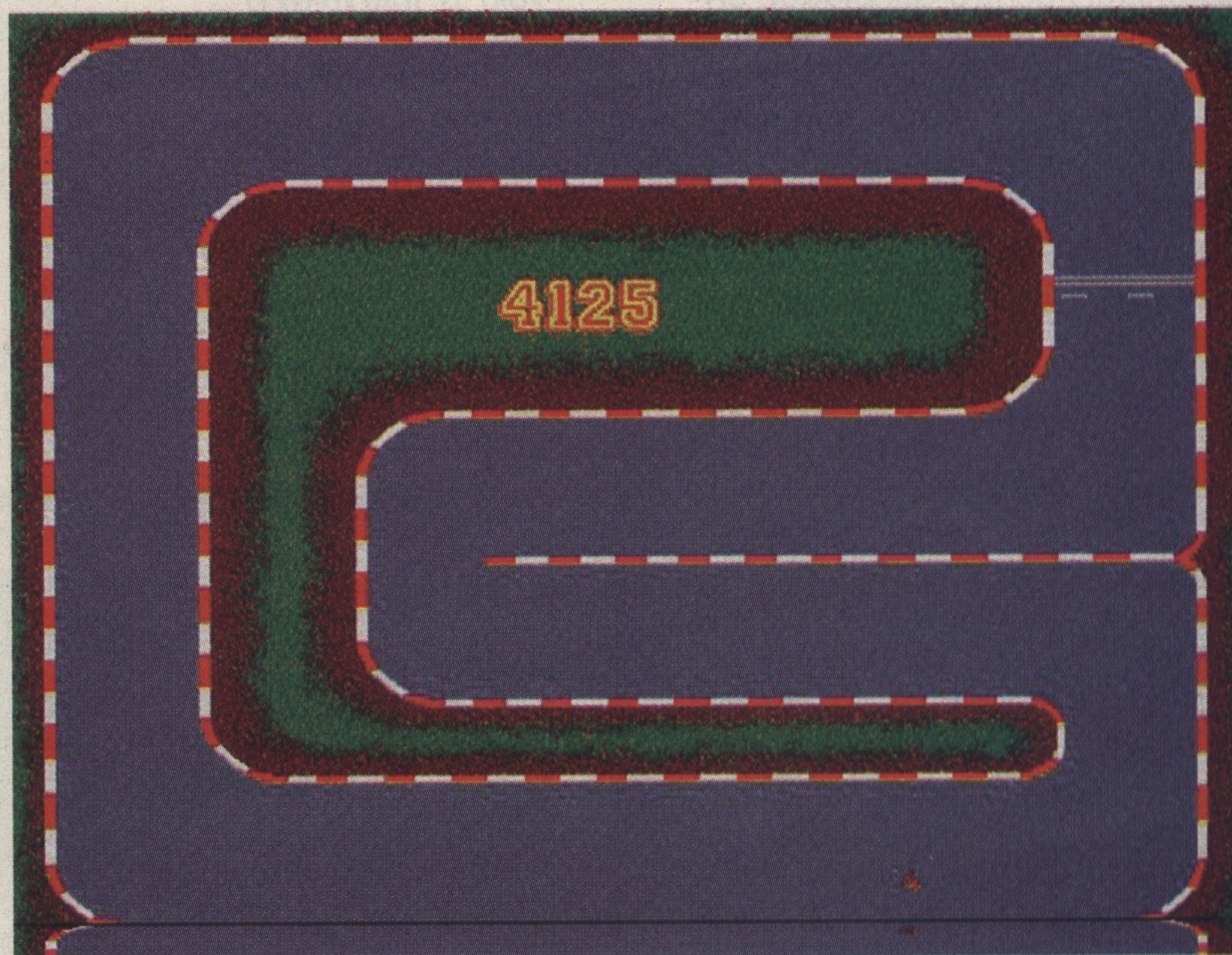
  if(agua==NULL) return;

  inc=(height-ALTURA-1)*100;

  for(x=0;x<ALTURA;x++) {
    memcpy(agua+x*wide,buf-
fer+(inc/100)*wide,wide);
    inc-=desp/10;
    desp+=dec;
    if(desp<2500) dec=1;
    if(desp>3500) dec=-1;
  }
  memset(agua,0,wide);
  memcpy(buffer+(height
ALTURA)*wide,agua,wide*ALTURA);
}

void __export divlibrary
(LIBRARY_PARAMS){}

void __export divmain
(COMMON_PARAMS) {
  AutoLoad();
  GLOBAL_IMPORT();
  DIV_export("post_process
_buffer",post_process_buffer);
}
void __export divend
(COMMON_PARAMS) {
  div_free(agua);
}
```



Otro efecto de agua sobre un juego de carreras.





Antes de analizarlo, expliquemos qué hacen y cómo se utilizan las funciones *memset* y *memcpy*. Veamos su sintaxis:

*Memset(void * destino, void * origen, int bytes);*
*Memcpy(void * destino, int dato, int bytes);*

Son funciones que copian un bloque de datos (origen) en un buffer denominado destino. El tamaño del bloque a copiar se indica en el parámetro bytes. Si el origen es una variable constante, se usará la función *memset* y éste se enviará mediante el parámetro dato. Si se trata de otro buffer o bloque de datos, debemos usar la función *memcpy*.

¿Por qué tratamos estas dos funciones? Pues porque la verdadera complejidad de este programa radica en las sentencias de copia de bufferes. Para empezar, el programa calcula el incremento de las líneas que vamos a utilizar para ver qué líneas debemos dibujar una sóla vez o ignorarlas. Una vez calculado el incremento, se utiliza en un bucle para ver cual de las líneas de la pantalla situadas sobre la zona de agua dibujamos. Cada vez que se ejecuta el bucle, copiaremos una línea completa sobre el buffer agua.

```
for(x=0;x<ALTURA;x++) {
    memcpy(agua+x*wide,
buffer+(inc/100)*wide,wide);
    inc-=desp/10;
    desp+=dec;
    if(desp<2500) dec=1;
    if(desp>3500) dec=-1;
}
```

La altura que tendrá nuestro efecto de agua se encuentra en la constante ALTURA. En función a esa constante, el bucle se ejecutará ALTURA veces, de forma que en cada paso calculemos una línea de nuestro efecto. Posteriormente se calcula el incremento a la siguiente línea a dibujar.

Una vez terminado, dibujamos una línea del color transparente (0 por defecto) que nos servirá de horizonte entre nuestro juego y el efecto. Para ello utilizamos la función *memset:*

```
memset(agua,0,wide);
```

Para finalizar con el procesamiento de este frame, copiamos el buffer donde hemos calculado el efecto (buffer *agua*) sobre zona inferior de la pantalla. Para ello recurrimos a la función *memcpy:*

```
memcpy(buffer+(height-
ALTURA)*wide,agua,wide*ALTURA);
```

Como se puede apreciar, utiliza el puntero buffer de la pantalla con un desplazamiento que corresponde al comienzo de la zona donde se debe empezar el volcado. Esto es todo. Tal vez lo más complicado radique en el análisis del cálculo del incremento, aunque no obstante, se limita a la aplicación de una función matemática parecida al cálculo del rebote de un objeto sobre una pared. Si no lo entendéis bien no os preocupéis. Podemos realizar cualquier tipo de función para calcular las líneas a dibujar.

Para los que quieran profundizar aún más en la programación de estas librerías de autocarga, les recomiendo encarecidamente que consulten el código de HBOY.DLL. Les aportarán muchas ideas nuevas, y seguramente les harán plantearse muchas incógnitas que poco a poco iremos desvelando.

Pues esto es todo lo que necesita para diseñar sus propias DLLs de autocarga y sus propios salvapantallas. Realmente se aplican los mismos conceptos que para las DLLs de funciones en cuanto a la programación de efectos visuales que en éstos. Ahora su único límite se encuentra en la imaginación. Prácticamente se puede hacer cualquier efecto que nos propongamos y convertirlo en un salvapantallas o en un DLLs de autocarga. Como todos los meses, les invito a realizar sus propios efectos, y si tienen alguna duda o sugerencia, no duden en escribir un e-mail a: trinidad@arrakis.es

Hacer efectos de agua será sencillo utilizando DIV

Pablo Trinidad



Direct X

Determinar los dispositivos

En el artículo anterior describimos las facultades y defectos de DirectX, la forma de configurar el compilador, etc. En este número desgranaremos el código del ejemplo que queríamos desarrollar para comenzar. De la teoría pasamos a la práctica.

Como ya hemos dicho en el número anterior describimos las facultades y defectos de *DirectX*. En este número desgranaremos el código del ejemplo que queríamos desarrollar para comenzar.

La inicialización de Direct X es un apartado realmente vital. Para ello utilizaremos un código simple

Como seguramente habréis comprobado, el ejemplo anterior es demasiado simple, prácticamente no hace nada. Pero en su sencillez está su fuerza. La inicialización de *DirectX* es tan primordial para nuestro proyecto, como puede serlo el sistema operativo para nuestra máquina. A continuación describiremos brevemente las partes diferenciadas de nuestro código que merecen un comentario.

Código del proyecto

En primer lugar, las declaraciones y variables globales.

```
#define INITGUID

#include <windows.h>
#include <windowsx.h>
#include <stdio.h>

#include "ddraw.h"

LPDIRECTDRAW   lpDD;
LPDIRECTDRAW2  lpDD2;

HWND hwnd;
HWND hlistbox;
```

Las librerías que incluimos nos dan acceso a las funciones básicas para crear nuestro proyecto, además de ello, incluimos la librería *ddraw.h*, que nos da acceso a las funciones de esta librería.

Función de enumeración

La función que describimos a continuación es la que nos proporcionara los datos de los dispositivos de vídeo presentes en el sistema. Esta función es llamada por *DirectDrawEnumerate* hasta que dejemos de devolver valores *DDENUMRET_OK*.

```
bool WINAPI EnumDDrawDevice
(GUID FAR *lpGUID, LPSTR
lpDDescription, LPSTR lpDName,
LPVOID lpContext)
```

Control de eventos

Como toda aplicación de Windows acostumbra, necesitamos controlar los eventos de nuestro programa. Tan sólo tenemos que preocuparnos por dos eventos, el de destrucción de nuestra ventana, es decir nuestro programa, y el que toma los valores que nos devuelve la ventana de diálogo que crearemos más tarde. Según la elección que hagamos crearemos nuestro objeto *DirectDraw*, aunque esto, por su importancia lo describiremos más tarde con más detalle.

```
LRESULT CALLBACK WindowProc
(HWND hwnd, unsigned Msg,
WPARAM wParam, LPARAM lParam)
```

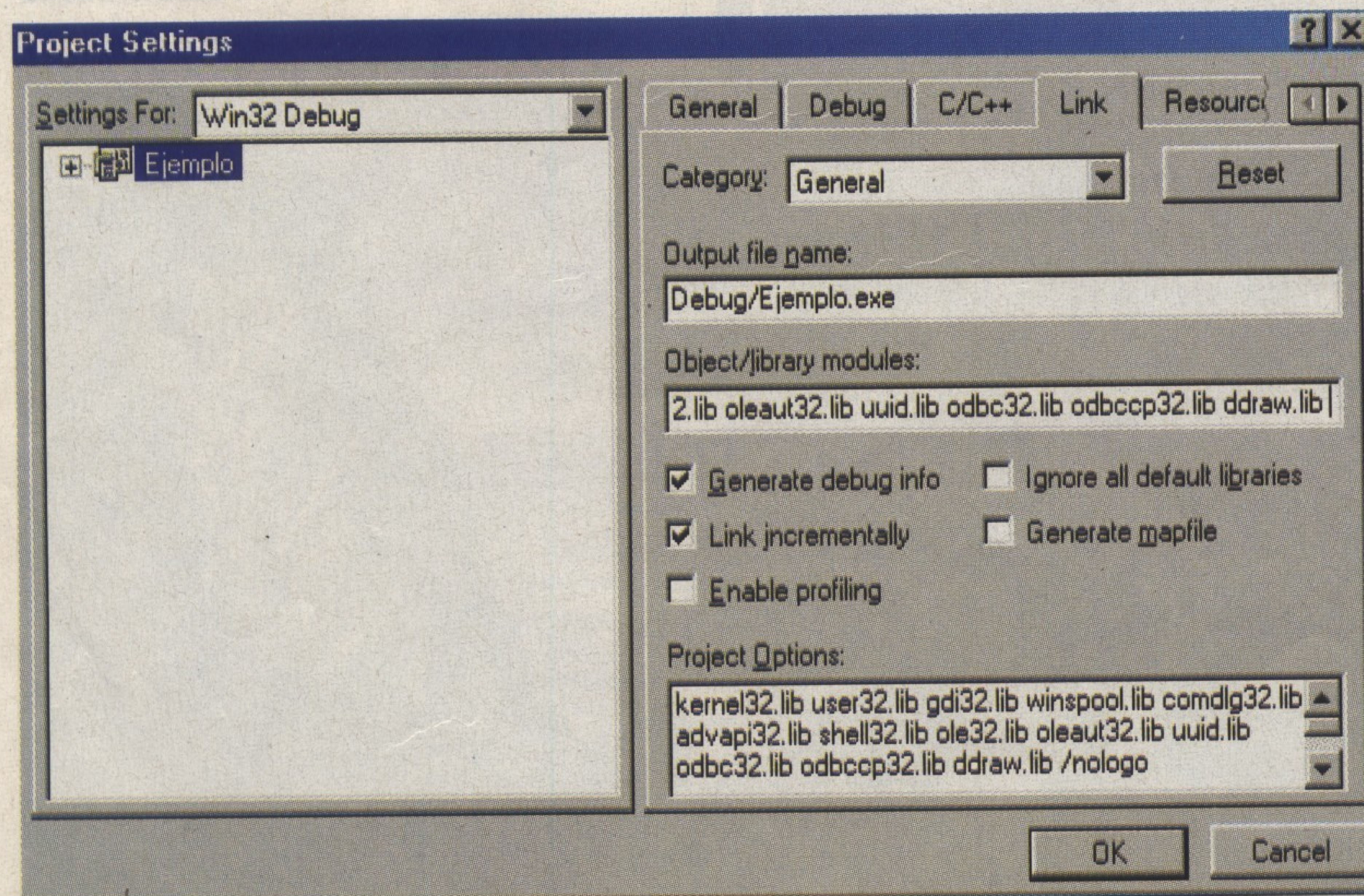
Función de inicialización

Una función de inicialización. El nombre es lo de menos, no nos dejemos engañar, lo único imprescindible es la función *WinMain*. Creamos nuestra clase, y nuestra ventana, así mismo, creamos la ventana de diálogo para poder mostrar al usuario los datos que nos devuelve *DirectDrawEnumerate*.

```
bool Init(HINSTANCE hInstance,
int nCmdShow)
```

WinMain

Esta función no tiene explicación, siempre y cuando hayamos programado antes bajo Windows.



No debemos olvidar la librería ddraw.lib entre las librerías necesarias para linkar nuestro ejemplo.

Llamamos a la función de inicialización, mostramos la ventana y enumeramos los dispositivos mediante *DirectDrawEnumerate*. Entramos en el bucle de control de mensajes, y cuando salimos, liberamos el objeto *DirectDraw*.

```
int PASCAL WinMain
(HINSTANCE hInstance, HINSTANCE
hPrevInstance, LPSTR lpCmdLine,
int nCmdShow)
```

Creación del objeto DirectDraw

Crear un objeto *DirectDraw* es tan sencillo como pasarle el identificador del dispositivo que vamos a utilizar, y un puntero al objeto *LPDIRECTDRAW* donde guardaremos la referencia que nos devuelve la función.

También podemos omitir la identificación del dispositivo, si le pasamos como valor *NULL*, la función nos devolverá un puntero al dispositivo por defecto, no es lo mejor, pero podemos hacerlo.

```
if (FAILED (DirectDrawCreate
(lpDevice, &lpDD, NULL)))
  {
    MessageBox(NULL, "No se
    pudo crear el objeto DDraw",
    "ERROR", MB_OK);
    return 0;
  }
```

A pesar de haber obtenido un puntero al objeto *DirectDraw*, y ser perfectamente válido para que lo usemos, no es el mejor. La función *DirectDrawCreate* nos proporciona la función primigenia, es decir, si queremos obtener las funciones de última generación de *DirectX*, debemos solicitar a la interfaz que acabamos de obtener que nos dé la última interfaz. Esta interfaz variará según la versión de *DirectX* con la que trabajemos. Una vez que obtenemos el puntero de la nueva interfaz, ya no necesitamos la antigua, por lo que la liberamos sin más.

```
// Obtenemos la interface correc-
ta de DDraw

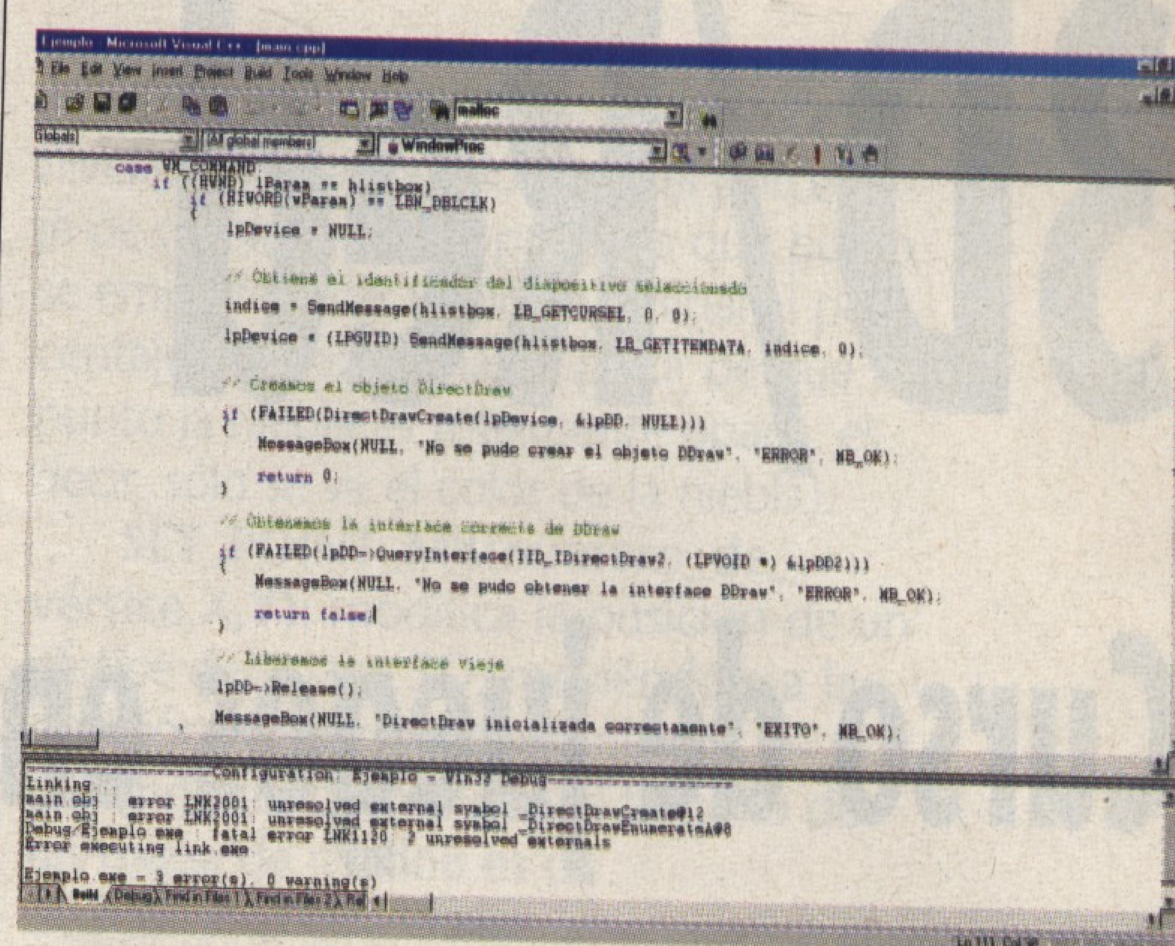
if (FAILED(lpDD->QueryInterface
(IID_IDirectDraw2, (LPVOID *)
&lpDD2)))
  {
    MessageBox(NULL,
    "No se pudo obtener la interface
    DDraw", "ERROR", MB_OK);
    return false;
  }

// Liberamos la interface vieja

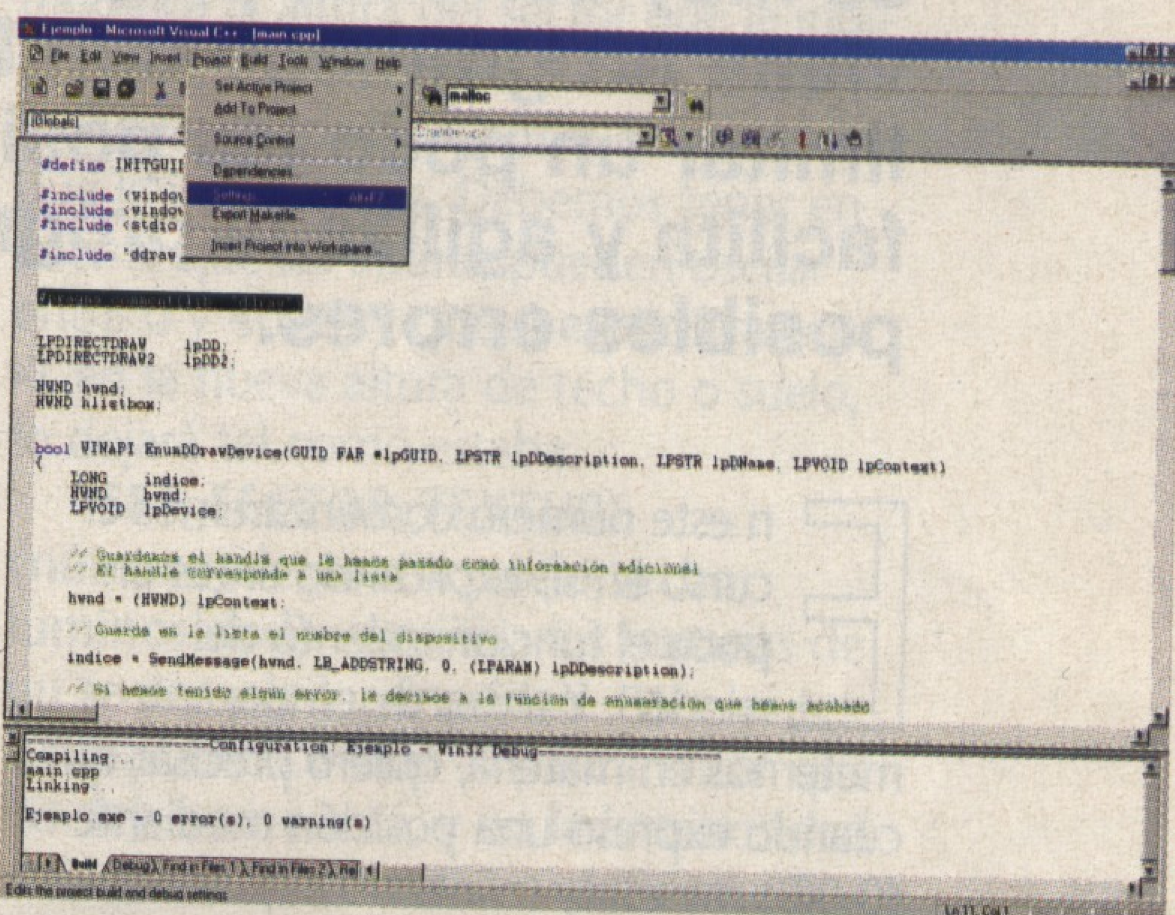
lpDD->Release();
```

Compilar el código

Para poder compilar el ejemplo debemos incluir entre las opciones del proyecto, la librería *ddraw.lib*, si la omitimos obtendremos una serie de errores de linkado. Aunque la opción más sencilla es incluir, entre las librerías de linkado de nuestro proyecto, aquellas que necesitemos, personalmente soy partidario de incluir directivas *pragma*. Esta orden nos permite proporcionar información al compilador de ciertas opciones de compilación, tal y como pueden ser, incluir librerías, de esta forma, en caso de que perdamos el fichero del proyecto, pero conservemos las fuentes, si en nuestro código hemos incluido las sentencias *pragma*, no tendremos problemas a la hora de compilar nuestro código, evitándonos la tediosa tarea de compilar y añadir una por una las librerías necesarias.



Si obtenemos estos errores de linkado, es síntoma inequívoco de que hemos omitido alguna librería.



Llegamos a las opciones de nuestro proyecto a través de Project, Settings.

La siguiente sentencia debemos incluirla al principio de nuestro código, si es que queremos evitar manipular las *settings* de nuestro proyecto.

```
#pragma comment(lib, "ddraw")
```

Conclusión

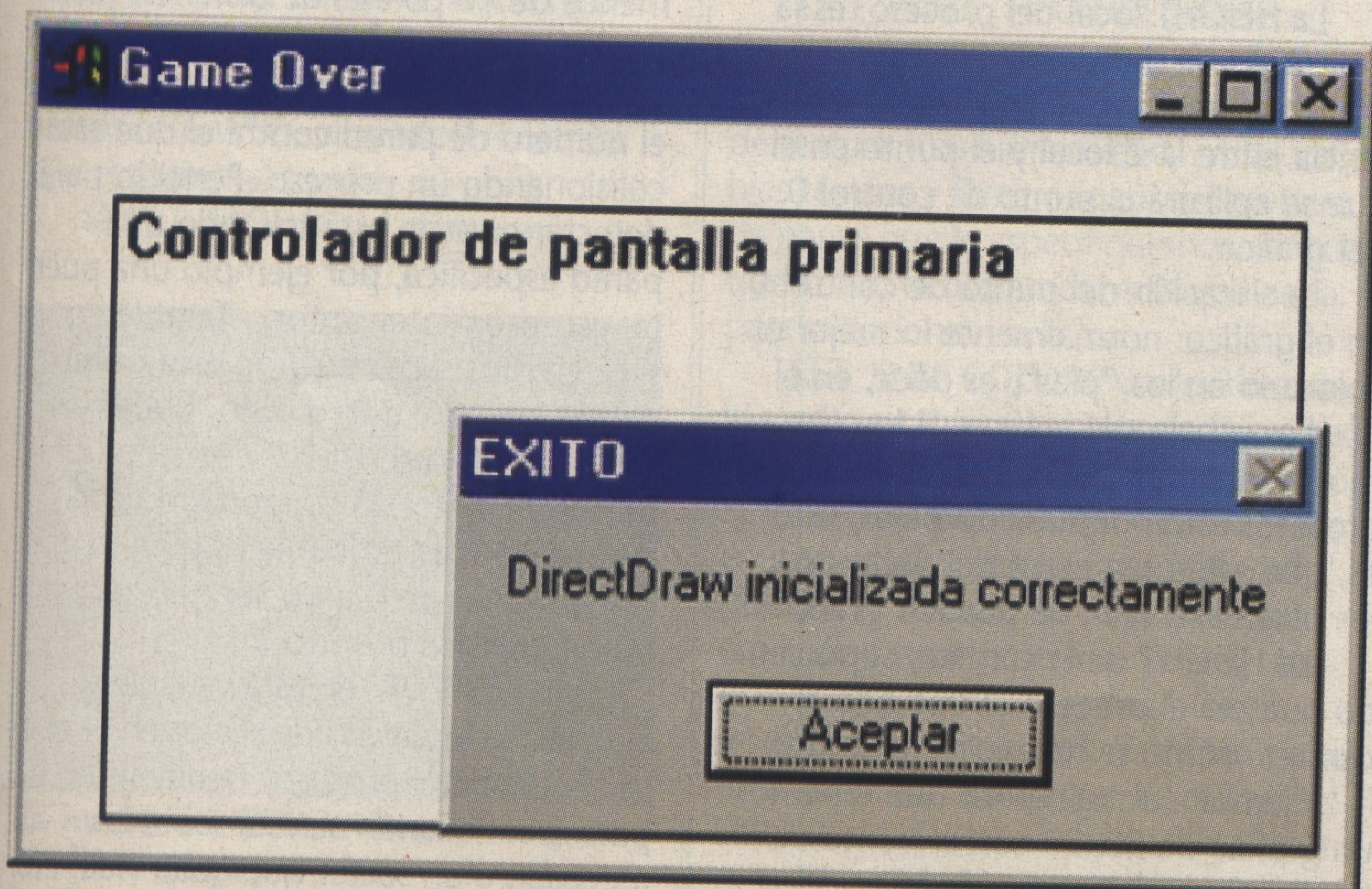
Creo que muchos lectores agradecerán la sencillez del código, y el no dar por conocidas cosas que algunos podrían no entender. En próximos números llevaremos nuestro ejemplo a nuevas metas, comentando en estas páginas sólo la parte que consideremos imprescindible, dejando el grueso del código para el CD.

> La utilización de un código sencillo y el conocimiento profundo de sus detalles nos dará mucha seguridad

Próximamente

En el siguiente articulo empezaremos a conocer las superficies. Estas estructuras son la clave de *DirectDraw*, puesto que son las que nos permiten mostrar la información en pantalla. Además prepararemos un ejemplo que nos permitirá cargar una imagen y mostrarla en nuestra ventana o a pantalla completa.

Armando Vélez



Este es el resultado de nuestro ejemplo, sencillo pero eficaz.

3D/Red

Curso de juegos en 3D: DIV Deathmaker (I)

La programación de juegos usando el modo8 es muy sencilla, ya que el engine general ya viene programado con Div. Quizá esto pueda limitar un poco las posibilidades pero también facilita y agiliza la programación, y salva de posibles errores.

En este número comenzaremos el curso en sí; explicando antes un poco el funcionamiento del modo8. Y antes de empezar a meternos en materia, quiero precisar que cuando expreso una posición mediante (0,0,0) me estoy refiriendo a las variables X, Y y Z.

La estructura M8

Existen 10 registros en esta estructura, desde m8[0] hasta m8[9], asignado cada uno al número de ventana de modo8 correspondiente. En la mayoría de los casos usaremos una sola ventana de modo8, con lo que no hará falta especificar; usaremos el registro m8[0], al que también podemos acceder poniendo sólo m8. Esto, siempre y cuando especifiquemos el número 0 de modo8 con start_mode8.

Esta estructura contiene cuatro variables propias de cada ventana de modo8:

Z: es la preferencia de impresión frente a otros procesos o ventanas de modo8. Se usa igual que la variable Z en dos dimensiones, y no se debe confundir con la Z local de cada proceso de modo8.

CAMERA: contiene el código identificador del proceso al que sigue la cámara. Si por ejemplo quisiéramos que la cámara siguiera al proceso Muñeco(); dentro de este proceso podríamos poner M8[0].CAMERA=ID; o bien desde otro proceso M8[0]. CAMERA=ID_DE_MUÑECO(); siempre teniendo en cuenta que Muñeco() debería ser un proceso de modo8, es decir, que sus coordenadas sean de modo8 (c_type=c_m8;).

HEIGHT: es la altura de la cámara respecto a la situación en Z del proceso al que sigue la cámara. Su valor predeterminado es 32. Es decir, si el proceso de la cámara está situado en (0,0,30) (X,Y,Z) y M8.HEIGHT vale 32, el punto exacto en el que está la cámara es (0,0,62).

ANGLE: es el ángulo vertical con el que mira la cámara, que inicialmente es 0 (centrado) y puede variar desde -128 (hacia abajo) a 128 (hacia arriba). Sin embargo, los valores límite no miran "totalmente" hacia arriba o hacia abajo, sino con un ángulo de 45° más o menos.

Las alturas

Quizá la parte más difícil de entender del modo8 son las alturas. En realidad hay cuatro factores que determinan la altura de un proceso respecto a la cámara:

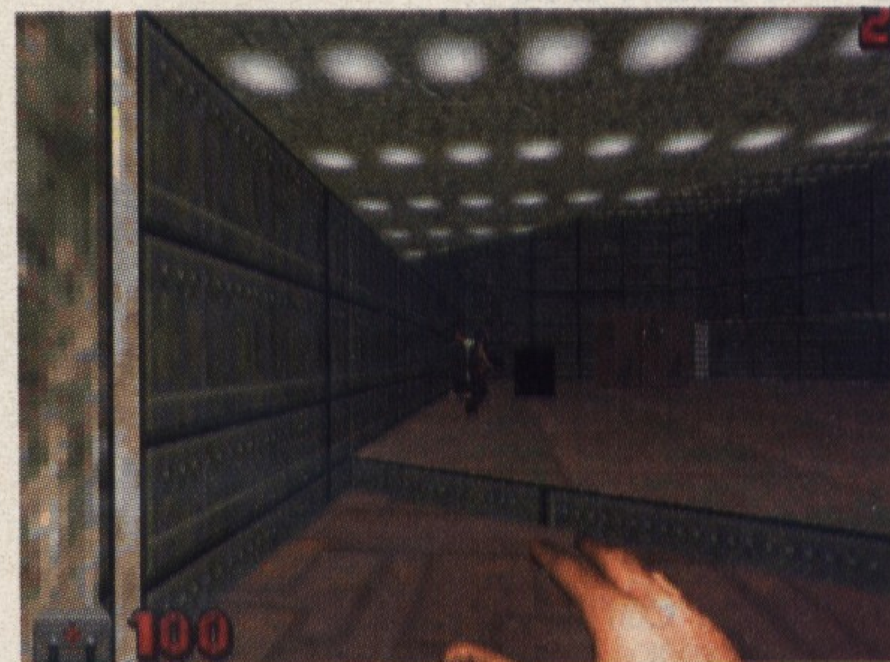
La Z local del proceso: esta coordenada es la altura exacta; junto con la X y la Y determina la situación exacta, el punto en el que se encuentra un proceso. Luego la posición del GRÁFICO vendrá determinada por otros factores.

La HEIGHT local del proceso: es la distancia que determina la altura del gráfico; para Div esta HEIGHT es la distancia entre la Z local y el punto en el que se aplicará el punto de control 0 del gráfico.

La situación del punto de control 0 en el gráfico: normalmente lo mejor es colocarlo en los "pies", es decir, en el píxel más bajo del gráfico. Si lo colocamos muy arriba puede resultar que los procesos estén demasiado bajos, esto es, que estén tan cerca del suelo que los píxeles se vean demasiado grandes.

La HEIGHT de la cámara, suponiendo que sea el proceso al que siga la cámara, como se ha explicado antes.

Ejemplo: imaginemos que tenemos un gráfico de 20x20 con el punto de control 0 en el centro (10x10), y lo situamos en (30,30,50) siendo su HEIGHT local 30. ¿Qué resultado tendremos? Pues el punto de aplicación del punto de control será el (30,30,80), y a partir de ese punto dibujará el gráfico, 10 píxeles hacia arriba y otros 10 hacia abajo. O sea, que el píxel más bajo nos quedaría en el (30,30,70).



El ZDOOM es una versión mejorada de Doom II que incluye BOTMATCH.

Variables locales de los procesos de modo8

Estas son las variables locales que tiene todo proceso que tenga coordenadas de modo8, es decir, que su variable local CTYPE sea igual a C_M8.

RADIUS: esta variable indica el radio de píxeles de grosor del proceso. Sirve para detectar colisiones con las paredes. Es decir, que si tenemos un proceso con un RADIUS de 10, ese proceso no se podrá acercar a una pared a menos de 10 píxeles de distancia (colisionará).

M8_WALL: esta útil variable guarda el número de pared contra el que está colisionando un proceso. Perfecto para detectar cuando está tocando una pared específica, por ejemplo una puerta, ascensor, interruptor... También por si queremos saber cuando está colisionando con una pared o no, podemos usar IF (M8_WALL0) y si es menor o mayor, es que está colisionando con alguna (lo más probable es que no valga menos de 0 a no ser que lo modifiquemos nosotros).

M8_SECTOR: en esta variable se almacena el número de sector en el que se encuentra el proceso. También útil para saber cuando se está sobre un ascensor, o un sector que quita vida, etc.

M8_NEXTSECTOR: indica el núme-

ro de sector al que está a punto de entrar el proceso. En otras palabras, si el proceso está colisionando con una pared y detrás de esa pared hay un sector, aquí se indica ese número de sector.

M8_STEP: es la cantidad máxima que puede "saltar" un proceso para acceder a un sector más elevado que el actual. Dicho de otro modo, el tamaño máximo del escalón que puede subir el proceso.

CNUMBER: es una variable que viene un poco camuflada, pero puede resultar bastante útil en juegos donde usemos más de una ventana de modo8 o modo7. Su uso es un poco difícil de explicar pero aún así es sencillo. Esta variable indica las ventanas en las que se debe mostrar el proceso. Si vale 0, se mostrará en todas las ventanas. Sin embargo, si queremos que se muestre sólo en algunas, debemos asignarle el valor de la suma de las variables C de cada ventana. ¿Qué son las variables C? pues una variable local para cada ventana de modo8 o modo7, que si por ejemplo es la ventana 5 pues será C_5, para la 2 C_2, etc. Entonces, si tuviéramos 3 ventanas (0,1 y 2) y quisiéramos que se viera el proceso sólo en la 1 y en la 2 pondríamos CNUMBER=C_1+C_2; y ya está. Posible aplicación: en un juego a pantalla partida si queremos que salga un puntero láser o indicador alrededor del enemigo cuando (por ejemplo) el lanzamisiles haya fijado el objetivo, no querremos que se vea este indicador en la pantalla del otro jugador. Por lo que usaremos CNUMBER para que sólo se vea en la ventana del jugador del lanzamisiles.

Funciones de modo8

Principalmente hay tres funciones necesarias para el uso del modo8, y son:

LOAD_WLD(archivo,fpg): obviamente, qué vamos a hacer si no cargamos el mapa 3D. Tenemos que especificar el nombre y ubicación del archivo WLD y la ID del archivo FPG, previamente cargado, donde se encuentran las texturas de este WLD. NOTA: sólo se puede cargar un WLD a la vez. Si cargamos otro WLD, el modo8 mostrará sólo el último cargado.

START_MODE8(id_cámara, númerom8,númeroregion): inicializa la ventana de modo8, teniendo que especificar el código identificador de la cámara (si todavía no se ha llamado al proceso que seguirá la cámara podéis poner ID con lo que usará el de el proceso que está ejecutando el START_MODE8), el número de ventana de modo8 (desde 0 hasta 9) y el número de región en la que se mostrará el modo8.

STOP_MODE8(númerom8): detiene la ventana de modo8 que le especifiquemos.

Hay varias funciones útiles, adicionales, que podemos usar en nuestros juegos con modo8. Son éstas:

GO_TO_FLAG(número_de_bandera): lleva al proceso hasta la posición exacta en la que se encuentra esa bandera. Las banderas las podremos situar con el editor de mapas 3D de Div2.

GET_POINT_M8(número_de_vértice,OFFSET X,OFFSET Y): devuelve, en las variables X e Y, las coordenadas de un vértice (en realidad una arista ya que no devuelve la Z) del mapa de modo8. Cuando "jugueteemos" con los vértices moviéndolos de un lado para otro, esta función será muy útil para saber en qué posición se encuentran después.

GET_SECTOR_HEIGHT(número_de_sector,OFFSET suelo,OFFSET techo): devuelve, en las variables suelo y techo, las alturas de suelo y techo del sector que le hemos indicado. También útil cuando estemos constantemente modificando estos parámetros para ascensores, puertas, etc.

GET_SECTOR_TEXTURE(número_de_sector,OFFSET suelo,OFFSET techo,OFFSET luminosidad): retorna, en las variables suelo, techo y luminosidad, los números de los gráficos de las texturas del suelo y del techo y la luminosidad con que se muestran. Casi siempre lo usaremos para obtener la luminosidad, para posibles cambios de luz que hagamos.

GET_WALL_TEXTURE(número_de_pared,OFFSET textura,OFFSET luminosidad): nos retorna el número de textura y su luminosidad, en las variables textura y luminosidad, del número de pared que le especifiquemos. Al igual que antes, lo usaremos, más que nada, para variaciones en la luz.

SET_ENV_COLOR(%rojo,%verde, %azul): establece el color de la niebla para el modo8, que llamaremos con SET_FOG. El color se debe establecer en porcentajes de rojo, verde y azul (RGB). Por defecto es (0,0,0) lo que equivale a negro, pero se puede cambiar constantemente para obtener resultados muy buenos. También debemos tener en cuenta que no sólo es el color de la niebla sino también el que se aplicará a la luminosidad de cada textura (0 será igual a este color, 15 la textura tal como es).

SET_FOG(inicio,final): establece el nivel de niebla, siendo inicio el porcentaje de niebla inicial (a partir de qué punto se empieza a ver la niebla) y final el porcentaje de niebla final (a partir de qué punto la niebla es totalmente opaca, es decir, sólo se ve el color de la niebla).

SET_POINT_M8(número_de_vértice,X,Y): modifica la posición de un vértice del mapa 3D, moviéndolo a la posición X e Y que le asignemos, teniendo en cuenta que el tamaño de un mapa de modo8 es de 30200x30200.

SET_SECTOR_HEIGHT(número_de_sector,suelo,techo): modifica la altura del sector que le indiquemos, situando su altura de suelo y techo en los parámetros suelo y techo que le indiquemos. Debemos tener en cuenta que las alturas pueden oscilar entre 0 y 4096, y que si especificamos -1 en la nueva altura de techo o suelo, lo dejará tal como estaba.

SET_SECTOR_TEXTURE(número_de_sector,suelo,techo,luminosidad): modifica las texturas de un sector y la luminosidad. Para las texturas podemos usar -1 para indicar que las deje como está, y en la luminosidad debemos especificar un valor entre 0 y 15, siendo 0 el color de ambiente, 15 la textura tal como es, y los valores intermedios distintos tonos de luminosidad, siendo -1 dejar la luminosidad tal como esté.

Es mejor no meter sectores dentro de otros sectores

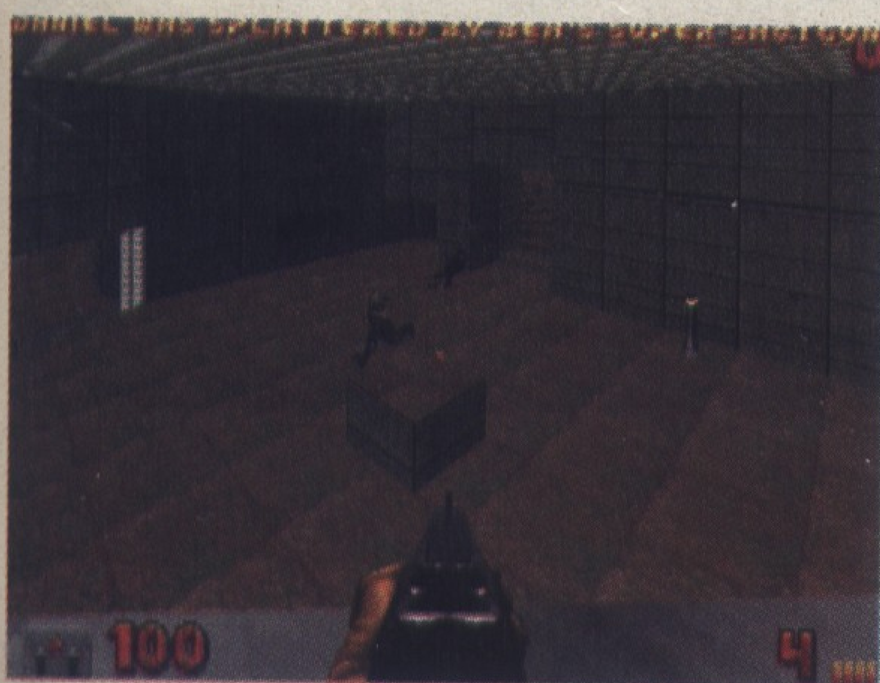
SET_WALL_TEXTURE(número_de_pared,textura,luminosidad): funciona exactamente igual que la anterior pero para una pared en vez de para un sector.

Los consejos de Ferminho

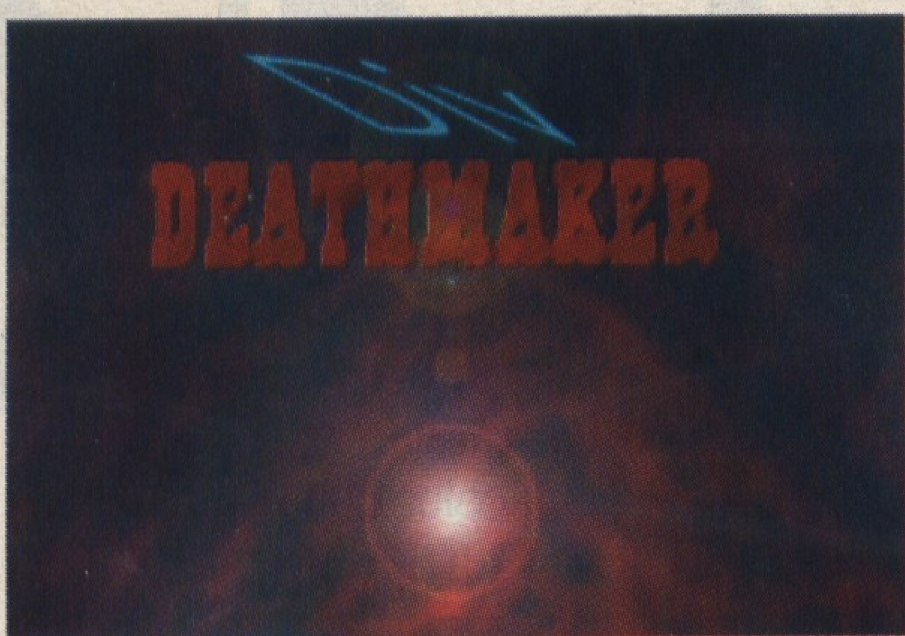
Para el diseño de los mapas, procuremos ser claros y no dejar todo como un barullo de líneas que se entrecruzan. Puesto que si no no nos daremos cuenta de posibles fallos en el mapa que luego harán que el juego no funcione bien, no detecte bien las colisiones o cualquier error posible. Para evitar errores, lo mejor es no meter sectores dentro de otros sectores (hay pequeñas probabilidades de que no funcionen bien), usar sectores más o menos "regulares" y no cruzar líneas nunca.

Antes de entrar en programación y tras haber terminado nuestro mapa 3D, es aconsejable apuntar, en un papel o en cualquier sitio para tenerlo a mano, los números de paredes y sectores especiales y qué acción debe ocurrir en ellos. Por ejemplo: WALL23=interruptor que abre wall30. SECTOR12: sector_lava resta 12hp/frame. Para luego no tener que estar mirando continuamente el mapa 3D al definir estas acciones mediante programación.

Si queremos un juego con una cali-



En nuestro BOTMATCH será imprescindible que aparezcan mensajes indicando muertes, suicidios...



La pantalla de título de nuestro juego "Div Deathmaker".


Unreal Tournament posee unos bots con una IA extremadamente elevada.


La sangre parece estar de moda en este tipo de juegos...

dad gráfica destacada, lo mejor es usar gráficos de personajes grandes, en un mapa grande con texturas grandes y SOBRE TODO una cámara con un RADIUS elevado. Así no se podrá acercar demasiado a las paredes y no se verán los píxeles muy grandes. Pero por supuesto, no se os ocurra ponerle un RADIUS muy grande a una cámara en un modo8 pequeño con gráficos de personajes pequeños, porque el resultado puede ser desastroso. El jugador se "frustrará" al no poder acercarse a las paredes (¿es que están recién pintadas o qué?).

El modo8 no es perfecto, bien, tiene algunos errores en la programación. ¿En qué se traduce esto? En posibles bloqueos o bugs internos durante la ejecución del programa. Pero no por ello debemos desesperarnos. Si nos fijamos bien en qué sector ocurre y qué sectores se veían en el horizonte al bloquearse, normalmente con reeditar estos sectores con más cuidado se soluciona el problema.

Vamos a crear un juego 3D con el modo8

Comenzamos nuestro juego

Ante el gran repertorio de posibles juegos en 3D que se pueden hacer, me he decantado para este curso por un BOTMATCH. Supongo que la gran mayoría sabréis lo que es. Por si acaso, os explico. Los BOTMATCH son normalmente opciones de juegos en 3D -como Zdoom o Unreal Tournament- en los que juegas en un escenario cerrado contra bots, con las mismas reglas que en un DEATHMATCH jugando en red contra otros jugadores humanos. ¿Y qué son los bots? Pues fácil, simplemente "emuladores" de jugadores humanos. Robots (de ahí bots) controlados por el ordenador que siguen un guión, con una IA (Inteligencia Artificial).

Pues a este juego le he llamado "Div Deathmaker" y nos va a servir para ver cómo crear un juego 3D con el modo8. Además, también podremos ver las principales características de la IA de los enemigos en un juego 3D (dado que en este tipo de juegos la IA es primordial).

En este número todavía no se incluyen los archivos del juego con el CD porque no vamos a meternos de momento en materia de programación (lo dejamos para el próximo número). Tenemos que tener planeado el juego, al menos por encima.

Primero, el juego va a permitir hasta 7 bots, que junto con el jugador ya son 8 (suficientes, si no el juego se ralentizaría en equipos lentos). Cada uno va a tener su configuración personal, pudiendo modificarla y guardarla en archivos para almacenar más de 7 perfiles de bots. El juego vendrá con un SKIN (conjunto de gráficos de personaje) base, pero luego se podrán añadir más para usarlos con los bots. También se podrá cambiar el escenario (con otros que se diseñen con el editor de Div2). Estos factores son imprescindibles para que la diversión perdure, ¿o no? :-)

Tenemos que tener claros los factores principales. Por ejemplo, va a ser sólo Deathmatch (no vamos a meternos en capturar la bandera y otros modos de juego) sin equipos (para no complicarlo), se va a poder establecer un límite de FRAGS (o puntos), se van a ir guardando las puntuaciones en los archivos de cada bot (y del personaje), los ítems reaparecen cada 100 frames tras haber sido recogidos...

Suponemos que todos los ítems serán de uso instantáneo, para no complicarlo más (no habrán ítems de almacenaje y uso posterior), y que cada jugador/bot comienza con vida 100 y blindaje 0 y sus valores máximos son 200 y 200. También que comiencen con la Qk22 y 30 balas.

También hay otros factores a tener en cuenta. Por ejemplo, qué armas e ítems van a estar disponibles en el juego. Aquí pongo una lista de los que habrá:

1. **Láser de impulsos:** munición ilimitada, lenta en disparar y poco potente.
2. **Qk22:** pistola de proyectiles estándar, no tiene mucha potencia y tiene que cambiar de cargador cada 10 disparos. Máxima munición de 150 disparos (15 cargadores). Para este arma tendremos que usar balas inmediatas.
3. **Ametralladora de combate:** usa munición de la Qk22, disparando a una velocidad superior y sin necesidad de recargar.
4. **Blaster de fusión:** dispara un láser aturdidor a una frecuencia baja. Usa células de energía (máximo: 300).
5. **Repetidor láser:** dispara células láser a toda velocidad. Es poco precisa. Usa células de energía.
6. **Lanzamisiles:** lento en recargar pero muy potente. Máximo de misiles: 20.
7. **Minas:** deja minas explosivas que explotan a los 3 segundos. Máximo: 10. Estas minas cuentan como ítem, porque para usarlas no necesitas arma sino directamente munición (las minas en sí).

Los ítems que habrá serán:

- ReGen pequeño: añade 10 puntos de vida.
- ReGen mediano: añade 25 puntos de vida.
- ReGen grande: añade 50 puntos de vida.
- Blindaje pequeño: añade 10 puntos de blindaje.
- Blindaje mediano: añade 25 puntos de blindaje.
- Blindaje grande: añade 50 puntos de blindaje.
- Cargador: munición para Qk22 y A.C. (1 cargador=10 disparos).
- Caja de cargadores: 5 cargadores.
- Célula compacta: contiene 5 células de energía.
- Batería: contiene 40 células de energía.
- Caja de misiles: contiene 5 misiles.
- Mina: una mina personal.
- Quad Damage: dura 10 segundos y permite infligir cuatro veces más daño en los enemigos.
- Caja explosiva: prueba a dispararle.

Esto es todo por este número, ya sabéis más o menos como va el modo8 y qué podemos hacer con él, y también cómo va a ser Div Deathmaker. En el próximo número nos meteremos en la programación.

Ferminho (Fermín Vicente)
ferminho@lycosmail.com


Como en el modo8 no se pueden insertar objetos 3D usaremos sprites.

El objeto personaje

Objetos protagonistas y secundarios

Hasta la más pequeña de las aventuras necesita de su protagonista, alguien que la realice, y de otros personajes, que le ayuden, que le entorpezcan e incluso algún malvado antagonista que haga todo lo que esté en su mano para evitar que nos salgamos con la nuestra.

Ya tenemos definido el guión de nuestra aventura, nuestro menú de opciones y las acciones entre objetos. Hasta hoy no hemos hablado de nuestro protagonista y del resto de personajes que componen nuestra aventura.

En nuestra primera aventura con *DIV*, tenemos un único protagonista, al que deberemos ayudar a resolver sus terribles problemas. Al mismo tiempo tenemos varios personajes, como nuestro compañero de habitación, el jardinero y los niños; que nosotros, como jugadores, no podemos controlar. Al definir los personajes ya tuvimos en cuenta su función en el juego, el papel que desempeñaban en nuestra aventura, etc. Por regla general, en todos los juegos de aventuras conviven el mismo tipo de personajes. Ya al principio veíamos que los juegos de aventuras y los juegos de rol tenían algunos aspectos en común, ya que al fin y al cabo, un juego de aventuras no era más que un juego de rol muy simplificado.

Tipos de personajes

Siempre que definamos personajes debemos distinguir entre personajes jugadores y personajes no jugadores. Un personaje jugador es aquel que controla el jugador de la videoaventura. Los personajes no jugadores son aquellos que no pueden ser controlados por el jugador, su comportamiento está predeterminado y están a merced del programador en el momento de la creación de la aventura.

En todo juego de aventuras siempre ha de existir, como mínimo, un personaje jugador. Este será el protagonista del guión que sigamos para la programación. Este personaje estará controlado por el jugador de la aventura, siendo éste quien decida que hará y cuándo. El programador, por su parte, tan sólo puede limitar las posibilidades del personaje, pero no su comportamiento. En nuestra primera aventura gráfica, limitamos la posibilidad de que el jugador entre en su habitación, a menos que tenga la llave de la puerta y la use con ella, pero cómo se hará esto depende por completo del jugador al controlar el personaje. Puede hacerlo en mucho o en poco tiempo, es más, si fuese algo más complejo, podría darse el caso de que varios jugadores resolviesen el puzzle de forma diferente. Por ejemplo, una vez dentro de la habitación, el jugador puede: abrir la cama, coger la cartera y hablar con el compañero; o hablar con el compañero, abrir la cama y coger la cartera; o también abrir la cama, hablar con el compañero y coger la cartera. Nosotros sólo limitaremos la posibilidad de que coja la cartera a menos que haya abierto antes la cama. Cuanto más variadas sean las posibilidades de resolver un mismo puzzle, aún cuando sólo haya una solución, más sencillo y entretenido será para el jugador, ya que si le obligamos a seguir un riguroso orden de acciones podríamos ponérselo bastante complicado.

Aunque como mínimo ha de haber un personaje jugador, puede ocurrir que haya varios personajes controlables por el jugador. Cada vez son más las aventuras gráficas en las que debemos usar varios personajes para resolverla. En la mayoría los personajes están determinados y debemos usarlos sin más, sin embargo, en otras como es el caso de *Maniac Mansion*, debemos elegir nuestro grupo de personajes, y la solución de la



¿Hablamos un rato o tiro de espada?



Este zombi es un individuo de "pocas palabras".



A veces las palabras no bastan.

aventura depende por completo de los personajes que hayamos escogido para jugar.

Nuestro sufrido jugador se encontrará a lo largo de la aventura que le estamos preparando diversas habitaciones que contendrán variados objetos. Además, debe encontrar a otros personajes de nuestra aventura, que le faciliten, le entorpezcan la solución o que simplemente decoren. Estos personajes que él no controla están a nuestra entera disposición. En nuestra aventura podríamos poner varios personajes decorativos, por ejemplo, varios estudiantes en la plaza de la residencia. Un gato juguetón en los alrededores del cobertizo del jardinero y cualquier cosa que se nos ocurra.

> Los personajes con los que el protagonista principal de la aventura debe interactuar son imprescindibles

Los PSI

Las siglas PSI corresponden a Personaje Seudo Inteligente. En artículos anteriores ya hemos trabajado con objetos. La primera vez que definimos un objeto fue la diana. Cuando mirábamos la diana recibíamos un mensaje y cambiaba su descripción. Podríamos decir que nuestra diana de ejemplo era un objeto seudo inteligente.

A diferencia de nuestro gato juguetón decorativo, con un PSI



Este viejecito parece una buena fuente de información.

podemos interactuar. En las primeras aventuras los personajes eran bastante simples. Cuando se entablaba conversación con ellos siempre se comportaban de la misma forma, dando una única respuesta y, cuando cumplían su cometido (su función principal en el juego) solían desaparecer de escena o quedar aletargados. Hoy, cada vez son más los personajes que varían su comportamiento durante el juego. En nuestra aventura tenemos un personaje que se comporta de esta manera: el jardinero. Nuestro jardinero simplemente nos dirá una frase suelta cada vez que intentemos hablar con él. Las frases las dará de forma aleatoria.

Cada vez es más común ver personajes que varían su comportamiento. En cualquier aventura reciente vemos como a un PSI no sólo responde cuando le hablamos o miremos, sino que podemos entablar conversaciones con él, e incluso puede variar su forma de actuar acorde a lo que haya pasado o deba pasar. Hay algunos personajes a los que incluso, de alguna forma, podemos inducirles a que hagan cosas por nosotros.

Indistintamente de lo complejo que pueda llegar a ser un PSI, las formas de interactuar con él pueden ser muy diversas. Puede ser de forma directa, como hablarle o darle un objeto; o de forma indirecta, por ejemplo, sacándoles de un apuro, como liberar a un preso de una celda. Aunque no es muy común apagar o abrir a un personaje, puesto que en las aventuras todo queda muy abierto, quién sabe si a alguien se le ocurre incluir a un PSI que sea una especie de

Cuadro 1

Acciones con el compañero de clase:

Ir hacia compañero	*Nos movemos hacia él*
Hablar con compañero	*Entablamos diálogo con él*
Mirar compañero	*Obtenemos una descripción de él*
Dar foto a compañero	*Finalizamos la aventura*
Dar lata de refrescos	*El compañero se bebe la lata y nos quedamos sin ella. Si ya le hemos dado una antes, la rechaza.*

Acciones con el jardinero:

Ir hacia jardinero	*Nos movemos hacia él*
Hablar con jardinero	*Simplemente nos dice una frase suelta*
Mirar jardinero	*Obtenemos una descripción de él*

Acciones con los niños aburridos:

Ir hacia niños	*Nos acercamos a ellos*
Hablar con compañero	*Entablamos diálogo con ellos*
Mirar compañero	*Obtenemos una descripción de ellos*
Dar pelota a niños	*Nos dan la foto*



Platicando con Jar Jar.

Cuadro 2

Acciones aplicables a un objeto	**Acciones aplicables a un PSI**
Ir hacia	*Ir hacia*
Mirar	*Mirar*
Mover	*Hablar*
Coger	
Abrir	
Cerrar	
Usar	
Dar	

robot al que podamos encender y abrirlo para repararlo. Puesto que todo depende de la imaginación del autor, la forma de interactuar con un PSI puede ser de lo más variada y original.

Nuestros PSI, además del jardinero, van a ser nuestro compañero de habitación y los niños aburridos. Nuestro compañero de habitación debe ser quién nos introduzca en el juego, y a él deberemos volver para resolver la aventura al final. Además, podríamos conseguir que durante el juego también pudiésemos hacer más cosas con él, como darle una lata de refresco o intentar animarle contándole un chiste. Respecto a los niños, los escucharemos como se burlan de la foto de la novia de nuestro compañero y deberemos dialogar con ellos para saber cómo conseguir la foto.

Con los PSI podemos hacer también cosas muy curiosas. Es muy posible que el jugador intente aporrear la puerta esperando que le abra su compañero. Podemos hacer un PSI que, cuando el jugador haya llamado varias veces a la puerta, salga de habitación de al lado y nos regañe por estar molestándole.



Aquí no parece haber nadie.

Implementar personajes no jugadores

Para realizar los personajes no jugadores vamos a tratarlos como si fuesen otro objeto cualquiera. De esta forma podremos realizar las mismas acciones sobre un personaje que sobre un objeto, definiendo en el propio personaje cuál será el resultado de las distintas acciones. En nuestra aventura podremos ir hacia un personaje, hablar con un personaje, mirar un personaje, usar un objeto con un personaje y dar un objeto a un personaje (se da/usa el objeto, no el personaje); no podremos mover, coger, abrir, cerrar, usar ni dar ningún personaje.



Acerquémonos a la gente.

Seguidamente definimos las acciones que vamos a realizar con cada personaje. Todo lo que no está especificado pasará bajo el control de nuestra función genérica de acciones (función "acciones"), que se encarga de dar los mensajes genéricos a las acciones comunes (ver cuadro 1).

Implementando el personaje jugador: El protagonista

Los personajes jugadores, los protagonistas de la aventura, no son como los PSI, un proceso fácilmente localizable en nuestro código fuente. Como ya hemos hecho con los objetos, el comportamiento de nuestro protagonista con cualquier otro personaje está definido en cada uno de ellos. Aún nos faltan un par de detalles para ultimar nuestro protagonista, como es hacerle hablar (mejorar la función que estamos usando de momento en nuestros ejemplos), hacerle visible y hacerle caminar por un escenario.

> Hay muchas formas de construir los diálogos, depende de lo que queramos y de nuestra imaginación

Para hacernos una idea más exacta, debemos pensar en los PSI como si otros objetos se tratase, ya que en realidad así es. La diferencia de llamar a uno objeto y al otro PSI es, en el momento de la programación, que la mayoría de las acciones a realizar con uno son excluyentes para otro.

El diálogo

Una de las formas más frecuentes, y más antiguas, de interaccionar con un personaje es a través del diálogo. En las aventuras gráficas el



Conversando plácidamente con un tipo bastante raro.

diálogo no juega un papel tan importante como en las aventuras conversacionales. A decir verdad, una aventura conversacional era un diálogo continuo hombre-programa. Aunque ha perdido relevancia aún juega un papel muy importante, tanto es así que el protagonista mantiene un contínuo monólogo para informar al jugador, dándonos descripciones u opiniones como: es demasiado pesado como para moverlo, no creo que eso funcione, etc.

Ya con los objetos hemos conseguido acercarnos a esta atmósfera de "monólogo", pero aún no hemos hecho nada sobre el diálogo. Para simplificar la programación es mucho más cómodo realizar la implementación de todo el diálogo desde el proceso del personaje con el que dialogamos. De esta forma, todo lo que podemos decir y todo lo que nos pueden decir estará, en el código fuente, en el interior del PSI.

Para el diálogo hay dos formas de enfrentarse. La más simple y que tiene mucha aceptación es el diálogo cerrado. Cuando un jugador decide hablar con un personaje, el diálogo se establece automáticamente y el jugador permanece a la "escucha" como un simple espectador. La otra forma es más compleja y puede llegar a serlo tanto como queramos. En la mayoría de las aventuras el jugador tiene la opción de indicarle al personaje que frase debe usar en cada momento del diálogo. En nuestra primera aventura gráfica con *DIV* usaremos esta última.

Para hacer que el diálogo con los personajes sea lo más ameno posible, debemos hacer que las secuencias de diálogo realizadas desaparezcan (si hemos hecho una pregunta, no volver a dar opción a hacerla) y, que si es necesario, generen nuevas (al hacer una pregunta recibimos una respuesta y, desde este momento, tenemos opción a preguntar sobre la respuesta. Tenéis un ejemplo del diálogo del protagonista con el compañero de clase cuando se encuentran la primera vez (se indican todas las opciones y está subrayada la opción escogida; a la derecha se indica el nivel de la conversación) (en el Cuadro 3).

En el próximo artículo mejoraremos la función habla y crearemos un PSI de prueba para ver una forma práctica de cómo implementar el dialogo con un personaje.

Miguel Adolfo Barroso



Vamos a ver qué quiere este tipo.

Cuadro 3: diálogo entre el protagonista y el compañero la primera vez que se encuentran:

Protagonista: *NIVEL 1*
- <u>La próxima vez no te des tanta prisa en abrirme la puerta.</u>
- ¿Te has quedado sordo o es que llamaba demasiado fuerte?
- ¿Porqué no me has abierto?
- Bueno, te dejo.

Compañero:
- No me he dado cuenta.

Protagonista: *NIVEL 1*
- <u>¿En qué piensas?</u>
- ¿Te has quedado sordo o es que llamaba demasiado fuerte?
- ¿Porqué no me has abierto?
- Bueno, te dejo.

Compañero:
- La he perdido...

Protagonista: *NIVEL 2*
- ¿El oído?
- Sea lo que sea, no lo has perdido al ir a abrirme.
- <u>¿Qué has perdido?</u>

Compañero:
- He perdido la foto de Filomena.

Protagonista: *NIVEL 3*
- ¿Y qué más da la foto?
- <u>¿Tan importante es esa foto?</u>
- Perder una foto tampoco es para tanto.

Compañero:
- No puedo vivir sin esa foto, la necesito

Protagonista:
- ¿Puedo ayudarte?
- ¿Te has quedado sordo o es que llamaba demasiado fuerte?
- ¿Porqué no me has abierto?
- Bueno, te dejo.



Empezamos a programar nuestro parser

Programación de Juegos de Estrategia - 5

En esta quinta parte del curso empezaremos a dar forma a nuestro *parser* (es decir, el editor de mapas). Cuando acabemos con este tema nos daremos cuenta de que podemos crear las fases de nuestro juego de una manera mucho más intuitiva (y rápida) que si las tuviésemos que programar desde cero.

Lo primero que tenemos que tratar es el entorno con el que nos tendremos que enfrentar a la hora de editar un mapa. Hagamos una lista de las cosas necesarias en la pantalla. Lo primero que necesitaremos es una región de la pantalla en la que se vea el mapa que estamos editando, de un tamaño suficiente para verlo bien, pero no tan grande que no se pueda poner nada mas en la pantalla (con unos dos tercios de la pantalla será suficiente). También necesitaremos botones para salir, guardar o recuperar la ultima versión, entre otros. El botón de salida tan solo cambia de estado el programa al menú principal. Los botones de guardar y restaurar salvan o recuperan los datos de los ficheros map_edit.dat y mob_edit.dat (sería mejor poder escribir el nombre del fichero por teclado, pero si no tenemos DIV 2 con DIV 1 no es posible leer de teclado). Para editar el mapa, será necesario algún mecanismo para seleccionar que es lo que queremos poner en el mapa. Para esto ponemos los botones MOB y SUELO, y la "mini-ventana" que muestra el tipo de suelo o mob seleccionado. Y ahora pasamos a contar cómo se han realizado cada uno de estos detalles.

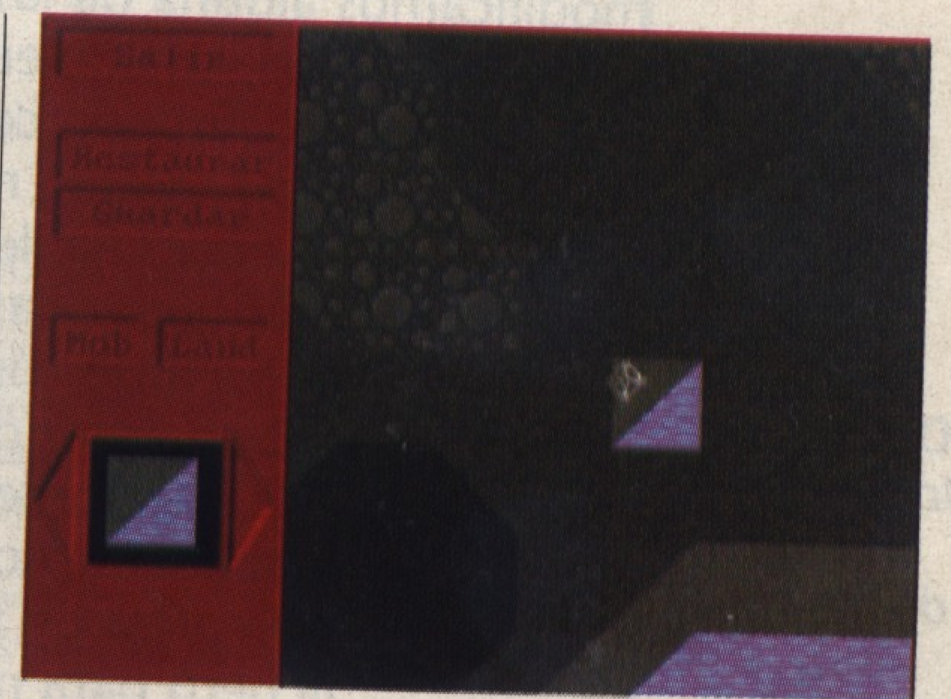
Hay que tener en cuenta si los mobs son voladores o no para comprobar si pueden estar en la posición del mapa que se les indique



Nuestro nuevo menú principal.

La región que muestra el mapa

La región en la que se muestra el mapa es igual que la que tiene el entorno de juego. Para obtener el tamaño de la misma, usamos los puntos de control que incluye DIV en sus gráficos. Con dos de esos puntos marcamos las esquinas Superior - Izquierda e Inferior - Derecha, y definimos una región con esos datos. Luego iniciamos un *scroll* dentro de esa región con un mapa grande en el que iremos pegando a modo de azulejos *(tiles)* los tipos de suelo. Estos suelos están en el mismo *.fpg* que el gráfico grande para poder hacer *map_put*. El *scroll*, como ya se comentó en artículos anteriores, se mueve por un proceso cámara que modifica las coordenadas del *scroll* de forma directa, en vez de ser éste el que sigue al proceso cámara. Hay que recordar que todos los procesos que deban aparecer en el scroll deben poner el *ctype a c_scroll*. También hay que tener el cuidado de transformar las coordenadas del ratón cuando se pulsa sobre la zona de *scroll*, ya que al moverse este, no hay relación entre las coordenadas de la pantalla y las de los procesos que se mueven por él.


Nuestro entorno de edición mostrando como poner un "tile" nuevo.

Cargando y Guardando

Como comentábamos al principio del artículo, tenemos un pequeño problema por usar DIV-1, y es debido a su carencia de funciones de manejo de *strings* (cadenas de caracteres). El problema tiene fácil solución con DIV-2, así que ya tenemos una razón más para actualizarnos. Debido a esa carencia, no le podemos pedir al usuario que introduzca el nombre del fichero por teclado, o generar nosotros uno. La solución que hemos seguido es limitar la edición a un solo fichero para mobs y otro para el mapa. Posteriormente deberá darse nombre al fichero desde el sistema operativo. Podría ocultarse esto mediante un sistema



Hemos puesto algunos mobs en el mapa.

similar al que hemos usado al cargar partidas guardadas, es decir, mediante un número determinado de "huecos" en los que "meter" los mapas creados, y se anima a los lectores a que lo intenten como ejercicio. Debido a que guardamos todos los datos de golpe tal y como están en la memoria, si modificamos alguna de las estructuras guardadas, cualquier fichero que estuviese guardado con el formato anterior se cargará mal, por lo que es muy aconsejable estar muy seguros de que todas las estructuras son como deben antes de ponernos a editar los mapas definitivos. Aquí vemos la ventaja de tener un fichero para mobs y otro para el mapa, ya que así la modificación de la estructura de los mobs no afecta a los ficheros de mapa, y viceversa, lo cual nos permite ahorrarnos la mitad del trabajo de reedición si hay que hacer alguna modificación en alguna de las estructuras.

Debemos recordar que todos los procesos que aparecen en el scroll deben poner el ctype a c_scroll

Selección mobs y suelos

Para seleccionar que es lo que queremos editar (el mapa o los mobs) tenemos los botones MOB y SUELO, que al pulsarlos cambian la muestra de la "mini-ventana" entre tipos de suelo y tipos de mobs. Si está seleccionado mobs, al pulsar las flechas de los lados de la ventana, se van mostrando los cinco tipos de mobs que hay, y si pulsamos sobre la región de scroll, se crea un mob que ocupa la casilla sobre la que se pulsó. Si está seleccionado suelo lo que se muestra al pulsar las flechas son los tipos de suelos que hay. Al pulsar sobre el mapa se cambia el tipo de suelo de esa casilla. Hay que tener en cuenta si los mobs son voladores o no para comprobar si pueden estar en la posición del mapa que se les indique. Lo mejor es que el propio proceso mob se ocupe de ello, es decir, si está en una casilla en la que no debe, se suicida. Los suelos, cuando los pones sobre el mapa, modifican la rejilla del mapa y la de juego, poniendo en una su índice en el *fpg* y en la otra un valor que indica si son tierra, mar o monte. Este último valor es el que se usa para saber si un mob puede estar en esa casilla o no, ya que los terrestres (no voladores) solo pueden ir por suelos del tipo tierra. Los valores de la rejilla de juego contienes dos informaciones, las centenas indican el tipo de mob, y las decenas y unidades, el tipo de suelo. Así, 101 significa suelo tipo tierra con un mob volador sobre él. Si fuese 2, significaría que no hay ningún mob y que el suelo es del tipo monte.

Y en el próximo número...

En el próximo número continuaremos con nuestro pequeño editor de fases.

Espero que nos leamos pronto.

Emilio Llamas Alba (YuMoK)

Código del proceso que nos indica si el ratón está sobre la región de juego o no.

```
PROCESS mouse_in_region()
BEGIN
  if( mouse.x < region_juego_x0 || mouse.x >= region_juego_x1 )
    return (false);
  end
  if( mouse.y < region_juego_y0 || mouse.y >= region_juego_y1 )
    return (false);
  end
  return(true);
END
```

Código del proceso que pinta constantemente un gráfico en las coordenadas indicadas por p_x y p_y. (Los datos son punteros, y se pueden modificar si se desea, pintándose el nuevo gráfico).

```
PROCESS Muestra_grafico(p_x,p_y,p_file,p_graph)
BEGIN
  graph=0;
  FRAME;
  LOOP
    x=*p_x;
    y=*p_y;
    file=*p_file;
    graph=*p_graph;
    FRAME;
  END
END
```

Otra forma de hacer la pausa:

Otra forma de hacer la pausa diferente a la usada hasta ahora por nuestro juego, consiste en guardar en una variable global el identificador del proceso principal, para que desde una función de pausa se pueda enviar una señal s_freeze_tree al mismo, e inmediatamente después enviarse una señal s_wakeup. La función de pausa quedaría algo así:

```
LOOP
  if(key(_P))
    Signal(id_proceso_principal, s_freeze_tree);
    Signal(id,s_wakeup);
    while (key(_P))
  FRAME;
    end
    while (!key(_P))
  FRAME;
    end
    Signal(id_proceso_principal, s_wakeup_tree);
  end
  FRAME;
END
```




Código del fragmento del programa principal dedicado al parser:

```
if(Estado_del_Programa==PARSER && !cambio_estado)
  fade_on();

  put(f_entorno,10,0,0);

  // Asignamos el gr fico del rat¢n
  mouse.graph=2;
  mouse.file=f_mouse;
  mouse.z=-10000;
  mouse.angle=0;

  /* Lanza los mobs y statics del editor y los gestiona*/
  Cargar_Bichos(FASE_EDITOR);
  Cargar_Mapa(FASE_EDITOR);  /* Carga el mapa para el edi-
  tor */

  start_scroll(0,f_tiles,1,0,1,0);
  camara_scroll(0,0,10);

  Pinta_Mapa(f_tiles);
  refresh_scroll(0);

  get_point(f_entorno,10,10,&x1,&y1);
  idtexto1=fun_button(f_entorno,101,x1,y1,&butt_exit);
  get_point(f_entorno,10,11,&x1,&y1);
  idtexto5=fun_button(f_entorno,105,x1,y1,&butt_load);
  get_point(f_entorno,10,12,&x1,&y1);
  idtexto2=fun_button(f_entorno,102,x1,y1,&butt_save);
  get_point(f_entorno,10,13,&x1,&y1);
  idtexto3=fun_button(f_entorno,103,x1,y1,&butt_mob);
  get_point(f_entorno,10,14,&x1,&y1);
  idtexto4=fun_button(f_entorno,104,x1,y1,&butt_suelo);
  get_point(f_entorno,10,15,&x1,&y1);
  idtexto6=fun_button(f_entorno,106,x1,y1,&butt_flecha_der);
  get_point(f_entorno,10,16,&x1,&y1);
  idtexto7=fun_button(f_entorno,107,x1,y1,&butt_flecha_izq);

  mob_o_suelo=SUELO_SELECTED;
  mob_actual=0;
  suelo_actual=0;
  grafico_escaparate=Valores_fpg_suelos[suelo_actual];
  fichero_escaparate=f_tiles;
  get_point(f_entorno,10,5,&x_escaparate,&y_escaparate);
    Muestra_Grafico(&x_escaparate, &y_escaparate,
      &fichero_escaparate, &grafico_escaparate);
    repeat
  FRAME;
until(!en_pausa);

while(Estado_del_Programa==PARSER)
  if(butt_flecha_der >> 2)
  //si mobs, incrementar variable mob_actual
  if(mob_o_suelo==MOB_SELECTED)
    mob_actual++;
    if(mob_actual>4) mob_actual=0; end
    grafico_escaparate=Valores_fpg_mobs[mob_actual];
    fichero_escaparate=f_mobs;
  end
  //si suelo, incrementar variable suelo_actual
  if(mob_o_suelo==SUELO_SELECTED)
    suelo_actual++;
    if(suelo_actual>20) suelo_actual=0; end
    grafico_escaparate=Valores_fpg_suelos[suelo_actual];
    fichero_escaparate=f_tiles;
  end
  FRAME(400);
end

  if(butt_flecha_izq >> 2)
  //si mobs, decrementar variable mob_actual
  if(mob_o_suelo==MOB_SELECTED)
    mob_actual—;
    if(mob_actual<0) mob_actual=4; end
    grafico_escaparate=Valores_fpg_mobs[mob_actual];
    fichero_escaparate=f_mobs;
  end
  //si suelo, decrementar variable suelo_actual
  if(mob_o_suelo==SUELO_SELECTED)
    suelo_actual—;
    if(suelo_actual<0) suelo_actual=20; end
    grafico_escaparate=Valores_fpg_suelos[suelo_actual];
    fichero_escaparate=f_tiles;
  end

  FRAME(400);
end

if(butt_mob >> 2)
  //poner el flag mob_o_suelo a mob
  mob_o_suelo=MOB_SELECTED;
  grafico_escaparate=Valores_fpg_mobs[mob_actual];
  fichero_escaparate=f_mobs;
end

if(butt_suelo >> 2)
  //poner el flag mob_o_suelo a suelo
  mob_o_suelo=SUELO_SELECTED;
  grafico_escaparate=Valores_fpg_suelos[suelo_actual];
  fichero_escaparate=f_tiles;
end

if(butt_load >> 2)
  Cargar_Bichos(FASE_EDITOR);
  Cargar_Mapa(FASE_EDITOR);
  refresh_scroll(0);
end

if(butt_save >> 2)
  Guardar_Bichos(FASE_EDITOR);
  Guardar_Mapa(FASE_EDITOR);
end

if(key(_ESC) || (butt_exit >> 2))
  //Pedir confirmaci¢n
  Estado_del_Programa=MENU_INICIO;
  cambio_estado=1;
  break;
end


if(mouse.left && mouse_in_region(1))
  x1=mouse.x;
  y1=mouse.y;
  if(Trunca_xy(&x1,&y1,1))
    x1/=X_TILE;
    y1/=Y_TILE;
    if(mob_o_suelo==SUELO_SELECTED)
      casilla=posicion_xy(x1,y1);
      rejilla_del_mapa[casilla]=grafico_escaparate;
      rejilla_de_juego[casilla]=TIERRA;
      if(grafico_escaparate==5)
        rejilla_de_juego[casilla]=MAR;
      end
      if(grafico_escaparate==6)
        rejilla_de_juego[casilla]=MONTE;
      end
    end
    if(mob_o_suelo==MOB_SELECTED)
  tipo_casilla=rejilla_de_juego[posicion_xy(x1,y1)];
  tipo_casilla=tipo_casilla/100;

  //si no hay mob (ni static) sobre el suelo.
  if(tipo_casilla == 0)
    //crear un mob nuevo
    num_id=find_mob();
    mobs[num_id].ocupado = 1;
    mobs[num_id].tipo=grafico_escaparate;
    mobs[num_id].casilla=posicion_xy(x1,y1);
    mobs[num_id].destino=mobs[num_id].casilla;
    switch(mobs[num_id].tipo)
      case MOSCA:
        mobs[num_id].vida=MOSCA_VIDA;
        mobs[num_id].flag_volador=true;
      end
      case ESCORPION:
        mobs[num_id].vida=ESCORPION_VIDA;
        mobs[num_id].flag_volador=false;
      end
      case WORKER:
        mobs[num_id].vida=WORKER_VIDA;
        mobs[num_id].flag_volador=false;
      end
      case TANK:
        mobs[num_id].vida=TANK_VIDA;
        mobs[num_id].flag_volador=false;
      end
      case SCOUT:
        mobs[num_id].vida=SCOUT_VIDA;
        mobs[num_id].flag_volador=false;
      end
    end

          mob(f_mobs,num_id);
      end
    end
    Pinta_Mapa(f_tiles);
    refresh_scroll(0);
  end
end
repeat
  FRAME;
until(!en_pausa);
end
```

El Francotirador

StarCraft

Un gran juego. Un gran guión. Un gran entorno. Una gran ambientación musical. En definitiva, un ejemplo a seguir. Gracias a las "pinceladas" de aventura gráfica que le han dado se consigue dar un efecto de continuidad a cada fase, cosa que no suele ocurrir en los juegos de estrategia.

El juego que nos ocupa puede parecer una versión futurista de su hermano el *Warcraft*, del que ya hablamos en el primer número. Ni mucho menos. Veamos las diferencias.



Menú Principal.

Gráficos y sonido
En el aspecto gráfico no ha cambiado demasiado, ya que se sigue usando la perspectiva isométrica para simular tridimensionalidad, pero se han cuidado mucho más los detalles como las animaciones en las fichas de las unidades o el mayor número de imágenes para hacer las animaciones (el giro de las torretas de los Terran hace que parezca un modelo 3D por la cantidad de posiciones que tiene). Otro detalle de los gráficos es la existencia de objetos que sólo sirven para dar ambientación al juego, como pueden ser los paneles solares que parecen ventiladores plantados a modo de árbol. No solo proporcionan una estética determinada a la fase en la que están, sino que también sirven para limitar la visibilidad. También el tema del sonido es un detalle que se cuida mucho. La música de fondo parece de película, cumpliendo su misión a la perfección: meter al jugador en el juego. Los efectos de metralleta, gritos, fuego, lasers, etc., hacen sentir como si estuviera ocurriendo una masacre en nuestro ordenador.

Inteligencia Artificial
Se puede decir que no se han olvidado de este punto, ya que posee una de las mejores IA's que se pueden ver en los juegos de estrategia hasta el momento. En las campañas no se ve realmente toda la inteligencia de la máquina, pero cuando se juega en una pantalla de las de *multiplayer* la cosa cambia. *StarCraft* tiene varios tipos de IA para las unidades, haciendo que actúen de forma semi-impredecible. Puede ocurrir que juegue al ataque de forma irracional, o de manera defensiva, o intentando acaparar todos los recursos de forma que el jugador se queda sin ellos rápidamente. Tiene múltiples patrones de acción, y se adapta a la situación en la que estén las defensas del jugador. Si el jugador tiene protegido su "espacio aéreo", el ordenador no intentara un ataque con naves, sino que enviará unidades terrestres.
Con respecto a la búsqueda de caminos, a diferencia de la que veíamos en el *Warcraft*, moverse por el mapa no es problema para las unidades en *StarCraft*. El algoritmo de búsqueda que utiliza debe ser uno de camino óptimo, ya que no solo encuentra una forma de llegar (cosa que no siempre conseguía el *Warcraft*), sino que encuentra el camino más corto hasta el destino.

Despedida
Concluimos aquí el análisis del *StarCraft*, comentando que una de las cosas que más atrae la primera vez que se juega a un juego son las animaciones entre fases, pero no hacen que se siga jugando después de la primera partida. Esto debemos conseguirlo mediante la IA, el sonido, los gráficos y si es posible, un gran argumento como el que tiene el *StarCraft*. Nos leemos...



El logo de este famoso juego de estrategia.



Introducción a la inteligencia artificial

Cómo programar un juego de rol

Para afrontar la programación de un buen juego de rol tenemos que dotar a los posibles enemigos del jugador de la suficiente inteligencia como para poner en apuros al más pintado. Eso se consigue programando una IA para dichos personajes.

El objetivo a largo plazo de la Inteligencia Artificial es alcanzar a la inteligencia humana, o incluso superarla, mediante métodos no naturales. Muchos investigadores de IA han ido diciendo que estábamos muy cerca de conseguir dicho objetivo.

1. Introducción a la Inteligencia Artificial

Los primeros estudios sobre Inteligencia Artificial fueron realizados por Alan Turing (ver Cuadro 1) hacia los años 30. Turing creía que hacia el año 2000 ya estaríamos hablando con ordenadores. Una de sus teorías más famosas fue la del "Test de Turing", mediante el cual se podía determinar si un ordenador era inteligente o no. El test consistía en someter a un interrogatorio a un ser humano y a una máquina, y si el interrogador era engañado por la maquina en un 50% o más podríamos definir a aquella máquina como el primer sistema inteligente creado por el hombre. Más tarde John Searle introdujo la Teoría de la "Habitación China", que ponía en duda el Test de Turing. Dicha teoría proponía la siguiente situación: se dejaba a un hombre en una habitación y se le iban pasando hojas con textos en chino (que él no entendía porque no conocía el idioma) y mediante unas pequeñas instrucciones que se le habían dado podía crear respuestas coherentes que podrían causar un error en el Test de Turing.

2. Tipos de IA

Dentro del campo de la Inteligencia Artificial podemos distinguir dos vertientes, que son:

- **Inteligencia Artificial Débil:** Los defensores de esta teoría dicen que se pueden crear programas que hagan tareas muy complejas pero que nunca serán verdaderamente inteligentes.
- **Inteligencia Artificial Fuerte:** Los científicos que defienden esta teoría afirman que sí se pueden conseguir comportamientos inteligentes.

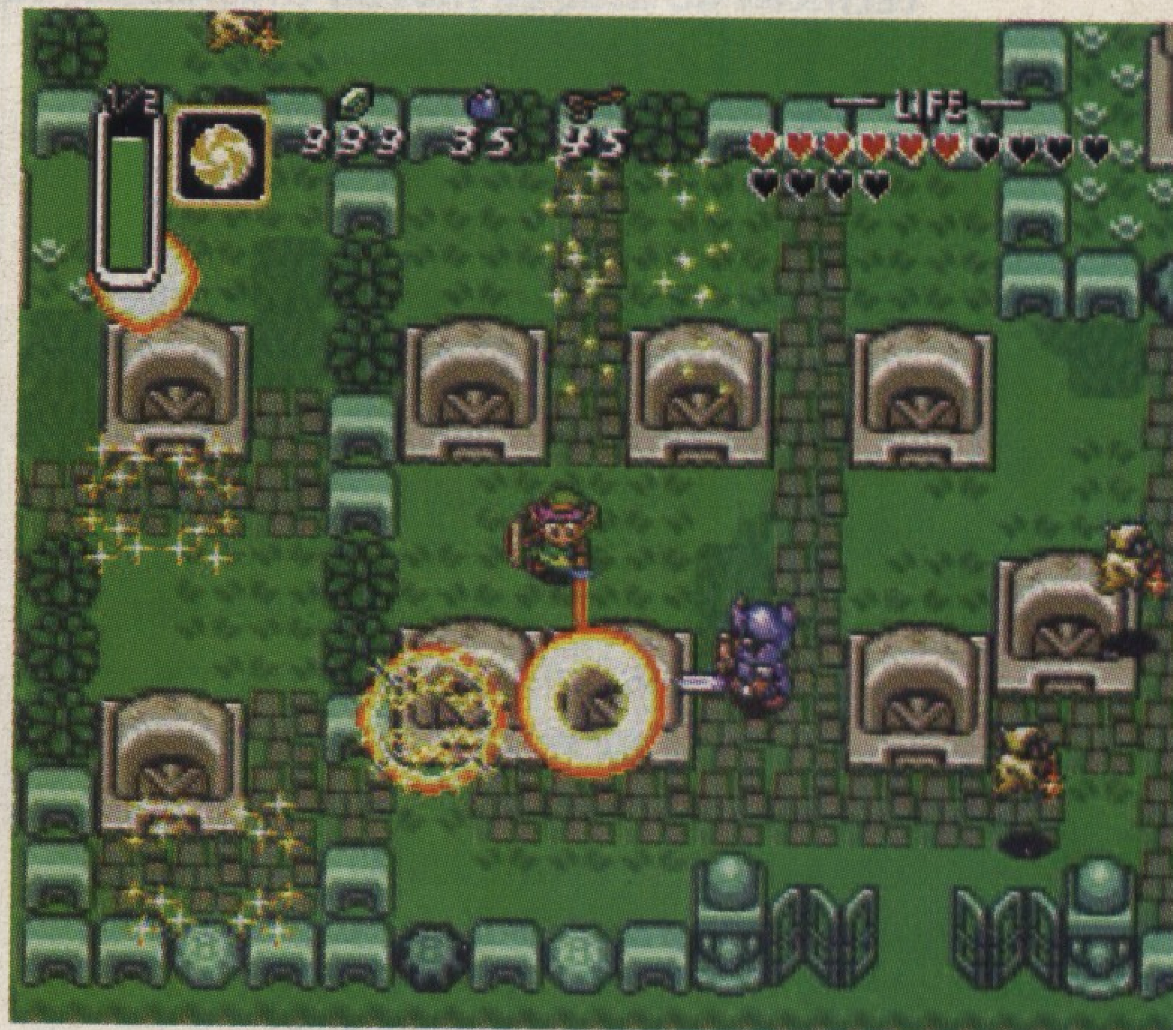
3. Métodos para conseguir IA

Desde los comienzos de la Inteligencia Artificial se han usado diversos métodos para conseguir resultados aparentemente inteligentes:

a) El método más simple es el de las **respuestas fijas.** Este algoritmo consiste en hacer lo que le han indicado, sin aprender, adaptarse o cambiar de comportamiento.

b) Otro método muy similar al anterior, pero con el que se obtienen unos resultados mucho mejores es el de las **reglas fijas**. En este caso tampoco se aprende ni se adapta, pero sí se reacciona a estímulos exteriores. Un ejemplo de esto serían los *Boids* de Craig Reynolds.

c) Un tercer método sería el de los **sistemas expertos**. Este simple método desarrollado en los 80 consistía en dar una solución a todos los problemas a los que se tendría que enfrentar la compu-



Zelda para Super Nintendo es un claro ejemplo del uso de waypoints para el movimiento de los enemigos.



Los enemigos de Zelda no destacan por su inteligencia.



Los enemigos en Diablo son algo más complejos y presentan comportamientos y técnicas de ataque mucho más elaboradas.

tadora. Esto se conseguía haciendo estructuras *if-else* enormes. Los dos defectos de este método son bastante claros. El primero es que para modificar las soluciones a todos los problemas es necesario que un hombre lo programe, y el segundo sería que el ordenador no podría hacer nada por solucionar un problema al que no se le ha indicado una solución.

En estos otros métodos ya no entramos en detalle ya que no serían propiamente IA, sino más bien Vida Artificial:

d) **Autómatas celulares:** Fueron definidos por John von Neumann (ver cuadro 2) hacía los años 40, basados en las máquinas de Turing.

e) **Redes neuronales.**

f) **Algoritmos genéticos**

4. Ejemplo

Ahora analizaremos un simple ejemplo de IA aplicado a lo que podría ser un enemigo en un RPG.

En el ejemplo (ver cuadro 3) se han implementado 9 modos distintos de IA. Cada modo esta definido con un número en la sentencia *switch(ia)*.

- **Caso 0:** Modo persecución.
  Este es el caso más simple, el enemigo perseguirá sin descanso al jugador. Para ello se buscan las coordenadas del jugador, y en función de éstas se decide a dónde se debe desplazar.
- **Caso 1:** Modo de detección estática por distancia.
  En este otro caso el enemigo permanece quieto hasta que el jugador se le aproxima a la distancia marcada (por defecto 150). La zona de detección se marca, aproximadamente, por una circunferencia. Una vez el enemigo lo ha detectado se activa el modo de IA 7.
- **Caso 2:** Modo de detección estática por visión.
  El enemigo permanece quieto en un punto, pero va cambiando la dirección a donde mira. Para detectar al enemigo se necesita que se cumplan dos condiciones: que esté a una distancia inferior a 150 y que coincida con el ángulo de visión que tiene el enemigo (la zona donde se detecta al enemigo será marcada, aproximadamente, por un gráfico rojo semitransparente). Una vez se le ha detectado se le persigue.
- **Caso 3:** Modo *waypoints* con detección por distancia.
  En este caso se usa la técnica de los *waypoints*. Ésta, consiste en marcar una serie de puntos por los que debe pasar el enemigo, de manera que siempre irá patrullando por un circuito cerrado definido por varios puntos. Si el enemigo, mientras patrulla,



En Diablo, los enemigos se mueven usando los waypoints con puntos aleatorios.

Cuadro 1. Biografía de Alan Turing.

Turing, Alan Mathison (1912-1954), matemático británico y pionero en la teoría del ordenador o computadora. Nació en Londres y estudió en las universidades de Cambridge y Princeton. En 1936, mientras era todavía un estudiante, publicó un ensayo titulado *On Computable Numbers (Sobre números calculables)*, con el que contribuyó a la lógica matemática al introducir el concepto teórico de un dispositivo de cálculo que hoy se conoce como la máquina de Turing. El concepto de esta máquina, que podría efectuar teóricamente cualquier cálculo matemático, fue importante en el desarrollo de las computadoras digitales. Turing también amplió su trabajo matemático al estudio de la inteligencia artificial y las formas biológicas. Propuso un método llamado el test de Turing para determinar si las máquinas podrían tener la capacidad de pensar. Durante la II Guerra Mundial trabajó como criptógrafo para el Foreign Office. Murió al administrarse un veneno, quizá por accidente, a la edad de 41 años.


Los enemigos de Baldur's Gate se mueven aleatoriamente por los escenarios.


En Baldur's Gate los enemigos no son tan inteligentes como esperaríamos, quizás porque la gracia del juego sea enfrentarse a otras personas a través de Internet.

Cuadro 2. Biografía de John von Neumann.

Neumann, John von (1903-1957), matemático estadounidense de origen húngaro que desarrolló la rama de las matemáticas conocida como teoría de juegos. Nació en Budapest y estudió en Zurich y en las universidades de Berlín y Budapest. Viajó a Estados Unidos en 1930 para incorporarse al claustro de la Universidad de Princeton. A partir de 1933 trabajó en el Instituto de Estudios Avanzados de Princeton (Nueva Jersey). Adquirió la nacionalidad estadounidense en 1937 y durante la II Guerra Mundial ejerció como asesor en el proyecto de la bomba atómica de Los Álamos. En marzo de 1955 fue nombrado miembro de la Comisión de Energía Atómica de los Estados Unidos.

Von Neumann fue un gran matemático. Destacó por sus aportaciones fundamentales a la teoría cuántica, especialmente el concepto de anillos de operadores (actualmente conocido como álgebra de Neumann) y también por su trabajo de iniciación de las matemáticas aplicadas, principalmente la estadística y el análisis numérico. También es conocido por el diseño de computadoras electrónicas de gran velocidad y en 1952 diseñó la primera computadora que utilizaba un programa archivado flexible, el *MANIAC I*. En 1956, la Comisión de Energía Atómica le concedió el premio Enrico Fermi por sus notables aportaciones a la teoría y al diseño de las computadoras electrónicas.

detecta a alguien a una distancia inferior a 150 lo persigue.

- **Caso 4:** Modo *waypoints* con detección por visión.
  Este caso sería igual que el anterior, salvo que para detectar al enemigo se usa el método usado en el caso 2.
- **Caso 5:** Modo punto aleatorio con detección por visión.
  El método aquí usado es muy similar al de los *waypoints*, salvo que los puntos por los que patrulla se generan de forma aleatoria cada vez. El método de detección que se usa aquí es el de detección por visión.
- **Caso 6:** Modo punto aleatorio con detección por distancia.
  Se usa el mismo método que en el caso anterior, salvo que la detección es por distancia y no por visión.
- **Caso 7:** Modo persecución con comprobación de distancia.
  El enemigo persigue al jugador, pero si este se le aleja mucho, deja de perseguirlo y vuelve a su modo de IA anterior.
- **Caso 8:** Modo alerta.
  Este modo se activa cuando se recibe una señal de alerta. Esta señal de alerta podría ser provocada al ver u oír algo, pero en nuestro caso podremos generar situaciones de alerta en puntos aleatorios con la tecla A.

Una vez se activa este modo el enemigo va a la zona donde se ha producido la emergencia. Una vez llega al lugar vuelve a su modo de IA anterior.

Ramón de España (MAQNAZILLER)

Cuadro 3. Código Fuente. *(principio)*

```
// PROGRMA PARA LA DIVMANIA Nº5
// RPG:IA

program ia1;

Global

    fpg1;     // VARIABLE PARA LA CARGA DEL FPG
    idjug;    // CODIGO ID DEL PROCESO JUGADOR
    idenem;   // CODIGO IDENTIFICADOR DEL ENEMIGO
    ia;       // MODO DE INTELIGENCIA ARTIFCIAL
    lastia;   // MODO DE IA ANTERIOR AL ACTUAL

Begin

    set_mode(m640x480);          // PONEMOS EL MODO GRAFICO
    fpg1=load_fpg("ia.fpg");     // CARGAMOS EL FPG DE LOS
                                 GRAFICOS
    idjug=jugador();             // LLAMAMOS AL PROCESO JUGADOR
                                 QUE ES EL PROCESO QUE
                                 CONTROLAMOS NOSOTROS
    idenem=enemigo();            // LLAMAMOS AL ENEMIGO

    ia=2;                        // MODO DE IA
    put(fpg1,30,320,240);        // PONEMOS EL FONDO

End


// PROCESO QUE GESTIONA AL JUGADOR QUE NOSOTROS
CONTROLAREMOS

process jugador()

Begin

    graph=1;     // ASIGNAMOS VALORES A LAS COORDENADAS
                 GRAFICO Y TAMA¥O...
    x=640/2;
    y=480/2;
    size=75;

Loop
```

Cuadro 3. Código Fuente. *(continuación)*

```
// CONTROL DEL JUGADOR

if(key(_up))
   graph=3;
   y-=2;
end

if(key(_down))
   graph=1;
   y+=2;
end

if(key(_right))
   graph=2;
   x+=2;
end

if(key(_left))
   graph=4;
   x-=2;
end

frame;

   End

End

// PROCESO QUE CONTROLA AL ENEMIGO Y
SU INTELIGENCIA

Process enemigo()

Private

ga;            // ANGULO ENTRE EL
               ENEMIGO Y EL JUGADOR
ep;            / POSICION DEL ENEMIGO
ec;            // CONTADOR PARA
               CAMBIO DE POSICION
mina=-45000;   // MINIMO ANGULO DE
               VISION
maxa=45000;    // MAXIMO ANGULO DE
               VISION
count;         // CONTADOR

   // ARRAY QUE MARCA LOS ANGULOS DE
VISION SEGUN LA POSICION EN QUE SE
ENCUENTRA EL ENEMIGO
   angles[3,1]= -450000,
 45000,
 135000,
 225000,

 45000,
 135000,
 225000,
 315000;

   // ESTRUCTURA PARA LOS PUNTOS DE
CONTROL
   struct waypoints[10]
x,y;
   end

   // ESTRUCTURA PARA EL PUNTO
ALEATORIO
   struct pal
x,y;
   end

   // ESTRUCTURA PARA EL PUNTO DE
ALERTA
   struct al
x,y;
   end

Begin
   // ASIGNAMOS VALORES A LAS VARIABLES
   graph=10;
   x=rand(0,640);
   y=rand(0,480);
   size=75;

   // MARCAMOS LAS CORDENADAS DE
CADA PUNTO
   waypoints[0].x=50;
   waypoints[0].y=50;

   waypoints[1].x=300;
   waypoints[1].y=50;

   waypoints[2].x=50;
   waypoints[2].y=220;

   waypoints[3].x=300;
   waypoints[3].y=220;

   Loop

// ACTUA SEGUN EL MODO DE IA QUE ESTE
PUESTO
switch(ia)

   case 0:

// MODO PERSECUCION
// SE MUEVE EN FUNCION DE LA POSICION
DEL JUGADOR A PERSEGUIR

// PERSIGUE LA POSICION DEL JUGADOR
if(idjug.x>x)
   graph=11;
   x++;
end

if(idjug.x<x)
   graph=13;
   x—;
end

if(idjug.y>y)
   graph=10;
   y++;
end

if(idjug.y<y)
   graph=12;
   y—;
end

   End

   case 1:

// MODO DETECCION ESTATICA
// PONE LA CIRCUNFERENCIA Y MIRA LA
DISTANCIA AL JUGADOR
// SI LA DISTANCIA ES MENOR A 150 SE
DETECTA Y SE PONE
// EL MODO DE IA 0

esf(x,y,200,maxa-45000,20);
   // PONE LA CIRCUNFERENCIA DE VISION
if(get_dist(idjug)<150)
   // SI ESTA CERCANO SE LE PERSIGUE

   lastia=ia;
   ia=7;

end

   End

   case 2:

// MODO VISTA ESTATICA
// CAMBIA DE DIRECCION CADA CIERTO
TIEMPO
// CAMBIA EL GRAFICO, PONE EL CONO Y
MIRA LA DISTANCIA AL
// JUGADOR. SI LA DISTANCIA ES MENOR A
150 Y SE ENCUENTRA
// EN EL ANGULO DE VISION SE PONE EL
MODO 0 DE IA

ec++;   // SE AUMENTA EL CONTADOR DE
POSICION

if(ec>100)  // EL CONTADOR A
            SOBREPASADO 100, SE CAMBIA
            DE POSICION
   ep=rand(0,3);  // POSICION ALEATORIA
   ec=0;
   mina=angles[ep,0];
      // SE CAMBIA LOS ANGULOS
      SEGUN POSICION
   maxa=angles[ep,1];
   switch(ep)  // SE ELIGE EL GRAFICO
               ADECUADO
case 0:graph=11;end
case 1:graph=12;end
case 2:graph=13;end
case 3:graph=10;end
   end

end

esf(x,y,100,maxa-45000,21); // SE PONE EL
CONO DE VISION

ga=fget_angle(x,y,idjug.x,idjug.y);  // SE
COGE EL ANGULO ENTRE JUGADOR Y
ENEMIGO
if(ga<-45000)ga+=360000;end  // SI ES
NECESARIO SE LE SUMA 360§ PARA QUE
COINCIDA CON LOS ANGULOS DE LA TABLA

if(get_dist(idjug)<150 && ga>mina &&
ga<maxa) // SI LA DISTANCIA ES MENOS DE
150 Y EL JUGADOR SE ENCUENTRA EN EL
CAMPO DE VISTA, SE LE PERSIGUE
   lastia=ia;
   ia=7;
end

   End
```

Modos y tonalidades con Fasttracker

Conceptos claves de armonía

Vamos a dedicar el artículo del número que nos ocupa a tratar una serie de trucos de teoría musical y armonía que esperamos que impulsen de una vez por todas el interés de los lectores, haciendo que se sientan *dominadores* de la música y no *dominados*.

Saber manejar un programa, no implica necesariamente obtener resultados satisfactorios con el mismo, y si no conseguimos resultados, lo más probable es que acabemos por hartarnos de él dejándolo a un lado. Como lo último que deseamos es que los lectores abandonen sus incursiones en el mundo musical por falta de recursos vamos a explicar unos cuantos conceptos claves de armonía que servirán para adentrarnos en el tema.

Ventajas del método que ahora utilizamos

Durante los tres últimos números hemos aprendido el uso de *Fasttracker 2* y con unas horas de dedicación, seguramente, habremos logrado ya algunos resultados que nos habrán sorprendido a nosotros mismos. Ya tendremos creado incluso algún "riff" que escuchamos una y otra vez sin cansarnos, y lo más posible es que más de algún lector piense: "pues no es tan difícil esto de hacer música". Y es que, con la ayuda de un buen secuenciador, nos ahorramos aprender lo que es, seguramente, una de las partes más tediosas de la música, la notación. Con el concepto de notación nos referimos al conjunto de signos que se utilizan para poder expresar de forma escrita las frases musicales que hemos ideado en ocasiones anteriores, con el fin de recordarlas o de que alguien más pueda interpretarlas.

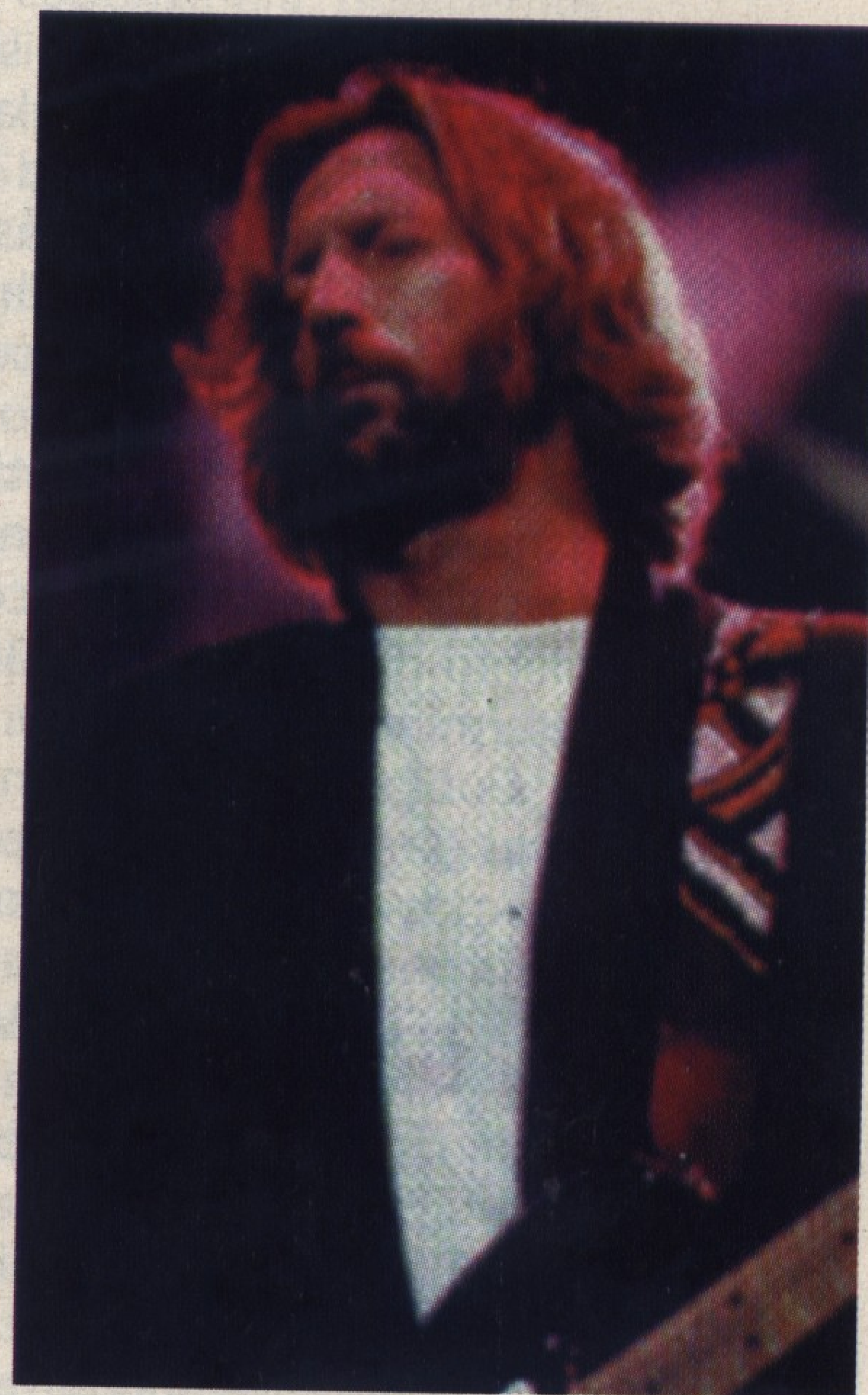
Es posible que sea necesario profundizar un poco más en los conceptos musicales para sacarle el rendimiento esperado a nuestro ordenador

Es mucho más rápido y cómodo para la composición escribir partituras que son interpretadas con infinidad de instrumentos. Cuando encontramos una nota que nos satisface no tenemos que buscar la línea del pentagrama a la que corresponde, sino que simplemente pulsaremos en modo de escritura la tecla asociada a esa nota, escribiéndose de esta manera en la partitura. También nos ahorramos tener que aprender la simbología que indica tiempos, ni tendremos que ir contando las notas que nos quedan hasta completar un compás. Bastará con que escribamos un dígito al principio del *pattern* que describirá la velocidad de reproducción de la misma. No necesitamos saber de *corcheas, semicorcheas, fusas, semifusas, puntillos* o *claves de sol*. Para nosotros la música se escribe con sólo ocho símbolos: "C", "D", "E", "F", "G", "A", "H" y por supuesto "#".

Para aquellos que quieran ir más allá

Como en esto de la música todo es subjetivo, habrá quien piense que no necesita saber nada más (de hecho hay muchos artistas que graban discos sabiendo muy poco de música). Pero también habrá lectores que hayan intentado hacer alguna versión de canciones conocidas y que habrán encontrado enormes dificultades para obtener todas las notas y detalles de determinadas partes de esas canciones.

La pregunta que surge ahora es, ¿cómo resolver este problema? ¿Existe realmente una manera rápida y efectiva para sacar todas las notas que componen una canción? ¿Se necesitan años de experiencia para sentirse suelto a la hora de transcribir temas musicales?. Pues bien, la experiencia es algo que no se puede pagar con dinero, y en el aspecto musical es algo esencial y muy deseable, pero con un método adecuado podremos ganar un tiempo considerable. Lo que sí está claro es que quien se haya dado de bruces con este problema necesita adquirir más conocimientos musicales, de eso no hay duda. Vamos a intentar explicar una serie de conceptos nada triviales que, una vez asimilados, nos harán ver la música de una manera bastante diferente: no como algo que surge exclusivamente de la inspiración, sino que puede ser planificado en muchas ocasiones con una exactitud casi científica. En mi modesta opinión, en un punto intermedio entre ambas maneras de tomar la música, se encuentra la mejor forma; la que posee la belleza del proceso creativo y la precisión del conocimiento. Sin más preámbulos vamos a introducirnos en el interesante mundo de las tonalidades y los modos griegos, ayudados, como siempre, de nuestro ya inseparable compañero *Fasttracker.*



Parte importante del secreto de los grandes guitarristas se basa en sus conocimientos de armonía.

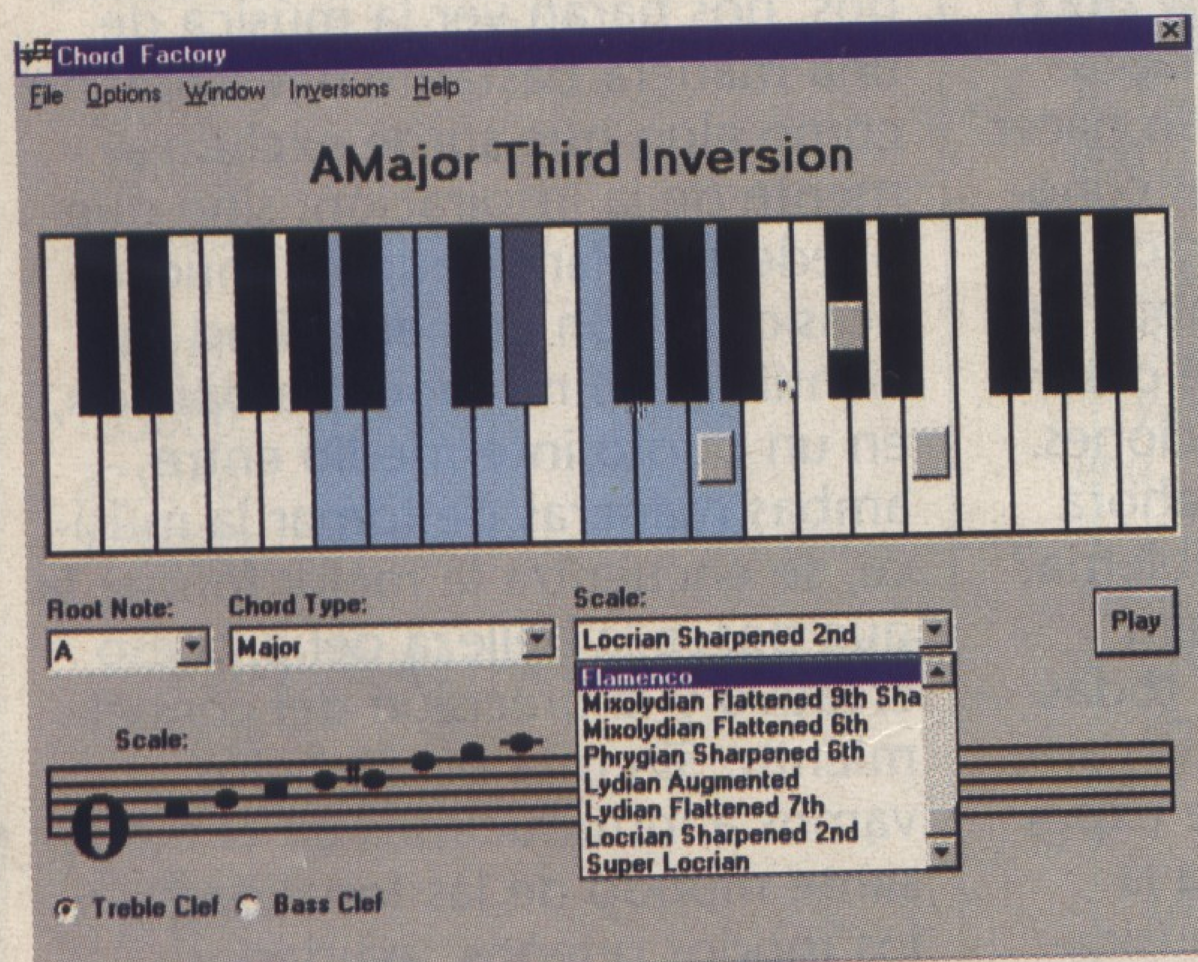
¿Para qué sirve exactamente saber de modos y de tonalidades?

Seguramente los lectores se habrán preguntado a menudo cómo es posible que su guitarrista favorito sea capaz de improvisar complicados solos sobre la música que está tocando su banda, sin equivocarse en ningún momento. Por supuesto, hay mucho de talento en esto, pero no todo es magia. Estos famosos músicos necesitan ser conscientes del "modo" para el que ha sido compuesta la música sobre la cual improvisan, para elegir las notas a tocar. Conocer la tonalidad o "modo" en el que se encuentra escrito un tema musical o fragmento servirá para saber de manera inamovible qué notas sonarán bien si las tocamos sobre el acompañamiento dado en un momento determinado y cuáles no.

Saber el modo en el que nos encontramos nos ayudará a saber qué notas podemos tocar y cuáles no

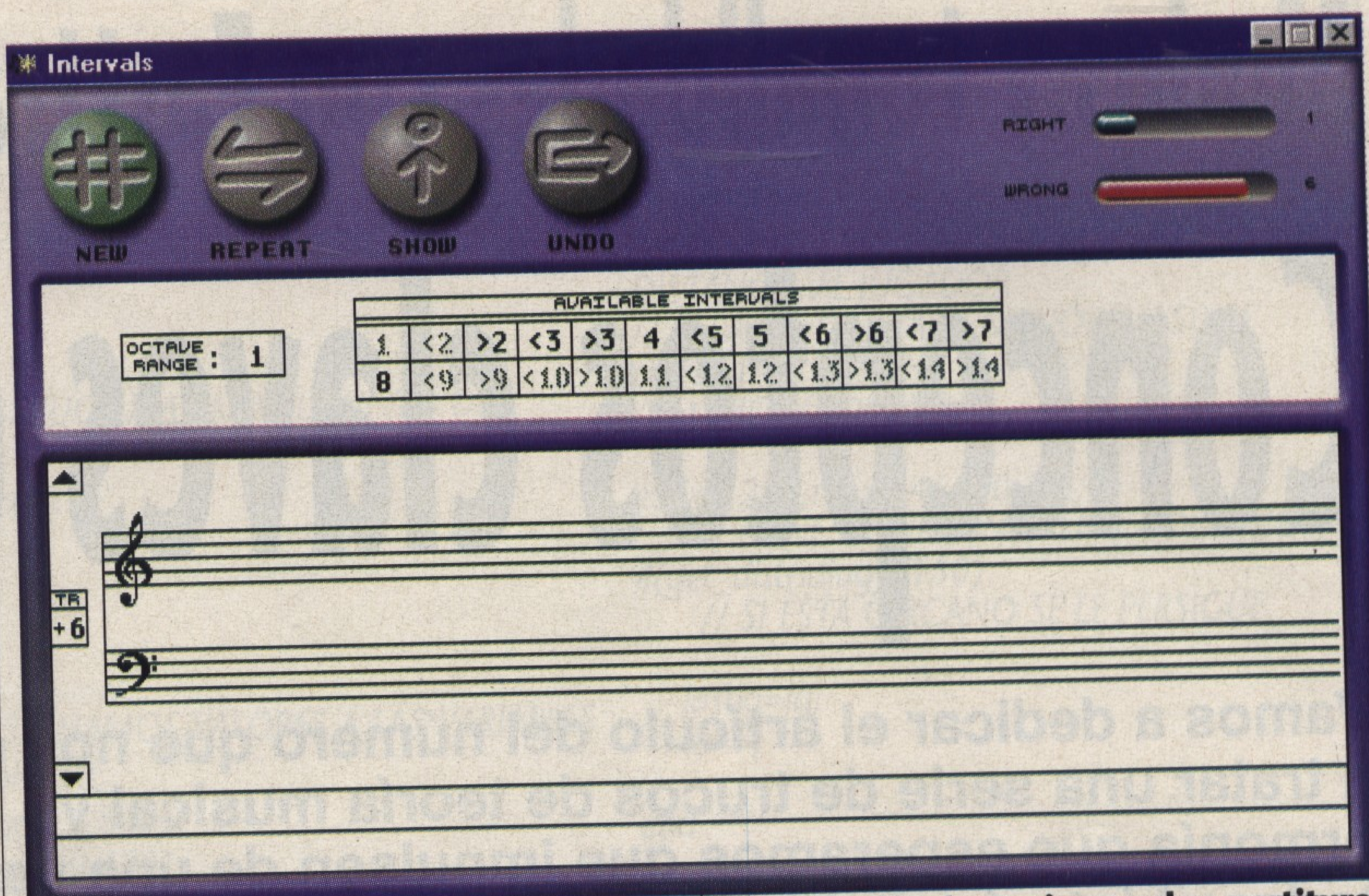
Contando con los tonos sostenidos (teclas negras del piano) tenemos doce posibles tonos o notas que son los siguientes: DO, DO#, RE, RE#, MI, FA, FA#, SOL, SOL#, LA, LA# y SI. Cada modo y tonalidad sólo permite tocar siete de estas notas, lo que reduce considerablemente las posibilidades entre las que tendremos que buscar a la hora de encontrar una nota apropiada para un determinado punto de nuestra canción.

Por fin llegamos a una definición más o menos formal de modo y tonalidad: digamos que el nombre que se le asigna a cada uno de estos conjuntos de 7 notas estará compuesto por un modo y una tonalidad. Con cada conjunto de 7 notas, dependiendo del orden en el que las toquemos, definiremos el modo, y la tonalidad vendrá definida por qué notas tocamos. Como la música está muy relacionada con algoritmos matemáticos, podremos estudiar con profundidad un caso determinado y luego explicaremos cómo hacerlo extensivo al resto.



Para cambiar de modo transportamos el patrón de intervalos hacia arriba o hacia abajo.

Tomaremos por ejemplo la tonalidad que posee las 7 notas blancas, es decir, todas las que no son tonos sostenidos: DO, RE, MI, FA, SOL, LA y SI. Definiremos sobre esta escala los siete modos griegos, "asociándolos" uno a uno con cada una de estas notas, en el orden siguiente: jónico (o mayor), dórico, frigio, lidio, mixolidio, eólico (o menor) y locrio. De esta manera, con las notas blancas hemos definido las siguientes configuraciones: DO mayor, RE dórico, MI frigio, FA lidio, SOL mixolidio, LA menor y SI locrio.



Gracias a los secuenciadores nos libramos de la complicada escritura sobre partitura.

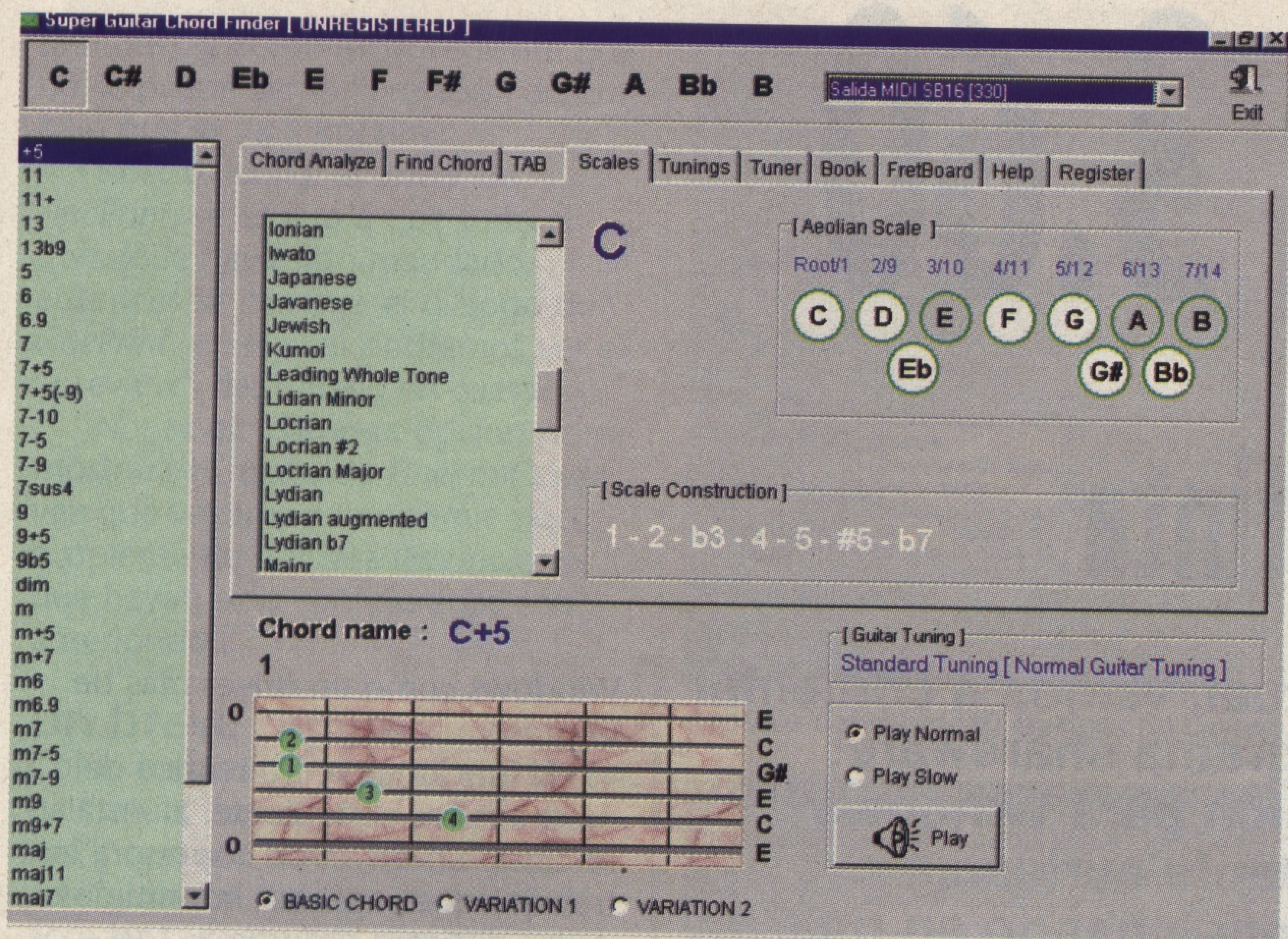
Los modos griegos

Lo único que sabemos hasta ahora es común a todos los modos y es que si componemos una canción en esta tonalidad, en cualquiera de estos modos, sólo podremos tocar las siete notas citadas. Lo que nos condiciona el modo es, como hemos dicho antes, el orden en el que las tocamos. Para que podamos entenderlo en pocas palabras, estaremos tocando en un modo determinado si la nota que tocamos más frecuentemente (o al menos aquella con la que comenzamos o acabamos los compases usualmente), es la que da nombre a ese modo. A esta nota la llamamos *tónica*. Dicho de otra forma, cuando decimos "asociar" un modo a una nota, significa a efectos prácticos, que podremos tocar aleatoriamente cualquiera de las 7 notas que corresponden a la tonalidad, pero deberemos hacer que predomine la nota que hemos asociado a nuestro modo (en el caso de Do mayor, por ejemplo, deberíamos tocar Do más frecuentemente que el resto de las notas).

Una alternativa que a menudo también funciona es la de tocar, en lugar de la tónica, la nota dominante, que será la resultante de incrementar en siete semitonos la tónica, por ejemplo, la dominante de LA, sería MI, la de DO sería SOL, etc... Hablando en términos de *Fasttracker*, sería como aplicarle a una nota siete veces la función de aumentar el tono en la sección *Transpose*. Vemos un ejemplo con notas de cómo centrarse en un modo determinado: supongamos que queremos componer algo en modo RE dórico. Tomamos algunas de las siete notas blancas para hacer un acompañamiento sencillo, por ejemplo RE, MI, LA y SI en este orden. Ponemos estas notas en la línea 0, 16, 32 y 48 respectivamente, reproducidas con un sonido de teclado, por ejemplo. Hacemos sonar este acompañamiento y habilitamos un canal para poder improvisar sobre él. Observamos que si tocamos DO, RE, MI, FA, SOL, LA o SI, nunca producimos sonidos disonantes, y además, si cada vez, que el compás vuelve a comenzar (suena el RE del acompañamiento), nosotros tocamos RE o su dominante que es LA, se produce un efecto extremadamente placentero para nuestros oídos. Sigamos improvisando hasta que alcancemos cierta destreza con el conjunto de teclas que corresponden a esta tonalidad y vayamos adquiriendo velocidad. En cuestión de cinco minutos estamos improvisando melodías tan válidas como la que más y, como se habrá podido comprobar, no ha resultado nada complicado.

¿Qué nos inspiran los modos?

Este es un buen momento para seguir probando todos los modos



Aún hay gran cantidad de escalas diferentes además de los modos griegos.

de manera que el lector decida por sí mismo que estilo musical le inspira cada uno de ellos. Una clasificación estándar podría ser la siguiente:

- El modo *jónico* o mayor es el más alegre de todos. Además es el más utilizado para muchos estilos musicales.
- El modo *dórico* puede inspirar desde blues a rock'n'roll pasando por la música celta o medieval, según el matiz que se le dé.
- El modo *frigio* es el que caracteriza al flamenco, aunque también puede sonar a música arábiga.
- El modo *lidio* tiene un tono muy misterioso. Es muy utilizado por el virtuoso guitarrista Steve Vai.
- El *mixolidio* encuentra su representación en el hard rock americano de los años 80, con grupos como White Snake, Bad Company o Def Leppard...
- El modo *eólico* o menor es junto con el jónico el más utilizado, pero su carácter es más triste y furioso. Infinidad de estilos musicales se basan exclusivamente en este modo, tales como muchas de las corrientes más significativas del Metal.
- Por último, el modo *locrio*, es posiblemente el que peor se acomoda a nuestro oído, y será utilizado mayormente en partes transitorias de las canciones, aunque estilos como el Grunge han hecho extenso uso de él.

¿Y el resto de las notas?

Ahora muchos lectores se preguntarán para qué rayos sirven entonces el resto de las notas (las que tienen sostenidos). Pues bien, para saber esto tenemos que introducir otro concepto, que es la modulación. Por modulación entendemos la acción de transportar en una o varias notas un número determinado de tonos hacia arriba o hacia abajo, haciéndolas sonar más graves, es decir cambiar la tonalidad. Dentro de las composiciones musicales se suelen hacer varias modulaciones, así como cambios de modo, lo que da riqueza a los temas. No es difícil detectar estas variaciones, ya que hacen cambiar bastante la intención de la música. Lo que ya puede no ser tan trivial es sacar cómo se ha hecho exactamente la modulación o a qué tonalidad hemos pasado exactamente.

Para llegar a saber estas cosas no tendremos más remedio que probar hasta encontrar la respuesta, aunque tampoco resultará extremadamente difícil si tenemos en cuenta lo que aquí se está explicando.

Entonces, para obtener otros modos tales como LA# lidio, bastará con tomar la secuencia que hemos estado utilizando hasta ahora (la de las siete notas blancas) y transportarla los tonos que sea necesario hacia arriba o hacia abajo. De esta manera, obtenemos otras tonalidades como por ejemplo las compuestas por las notas SOL, LA, SI, DO, RE, MI, FA# o FA, SOL, LA, LA#, DO, RE, MI, que serán nombradas por sus modos mayor y menor. En este caso serían, por ejemplo, SOL mayor-MI menor y FA mayor-RE menor, respectivamente.

Para acabar

Es posible que aún habiendo visto los modos y tonalidades más comunes, el lector llegara a encontrarse con canciones escritas con tonalidades compuestas por ocho notas en lugar de siete, o incluso seis. Existen así mismo escalas mixtas, indias, cromáticas, armónicas, que ofrecen otro tipo de sonidos, la mayoría más psicodélicos, y algunas veces son sencillamente variaciones de las modos que hemos visto. Con los modos griegos hemos cubierto un amplio abanico de posibilidades, y seguramente será más que suficiente para muchos de los lectores, que a partir de ahora aprovecharán mejor su tiempo en las sesiones de *Fasttracker* que realicen. En cualquier caso, sabemos bastante más que antes sobre armonía, lo cual quizá no sea mucho, pero tampoco es poco.

Transportando el patrón de intervalos de Do M-La m podremos generar cualquier tonalidad y modo

Sergio Cánovas

Modos griegos (en Do M - La m)

Nombre	Grado	Tónica	Carácter
Jónico (Mayor)	1º	DO	Alegre
Dórico	2º	RE	Blues, Medieval
Frigio	3º	MI	Flamenco, Áraba
Lidio	4º	FA	Sombrío
Mixolidio	5º	SOL	Rock americano
Eólico (Menor)	6º	LA	Triste, furioso
Locrio	7º	SI	Transiciones

FinePrint 3.60

Ahorrando papel y tinta

Como viene siendo habitual, vamos a comentar en esta sección otro programa shareware: FinePrint 3.60 en su versión para Windows 95/98. Su utilidad es bastante curiosa, consigue imprimir redimensionando en una sola página 2, 4 y hasta 8 páginas de un documento, lo que supone un gran ahorro de papel y tinta a la hora de imprimir grandes documentos.

FinePrint recoge el trabajo de impresión cuando éste es mandado a la impresora y tras redimensionar las páginas a la mitad, una cuarta parte o una octava, pone 2, 4 u 8 en una sola página y manda el trabajo a la impresora elegida. Lo curioso de este programa es que se instala como un driver de impresora, por lo que lo único que tenemos que hacer para utilizarlo es seleccionar este driver como impresora predeterminada.

Con FinePrint ahorraremos papel, tinta y tiempo, además FinePrint es muy fácil de usar

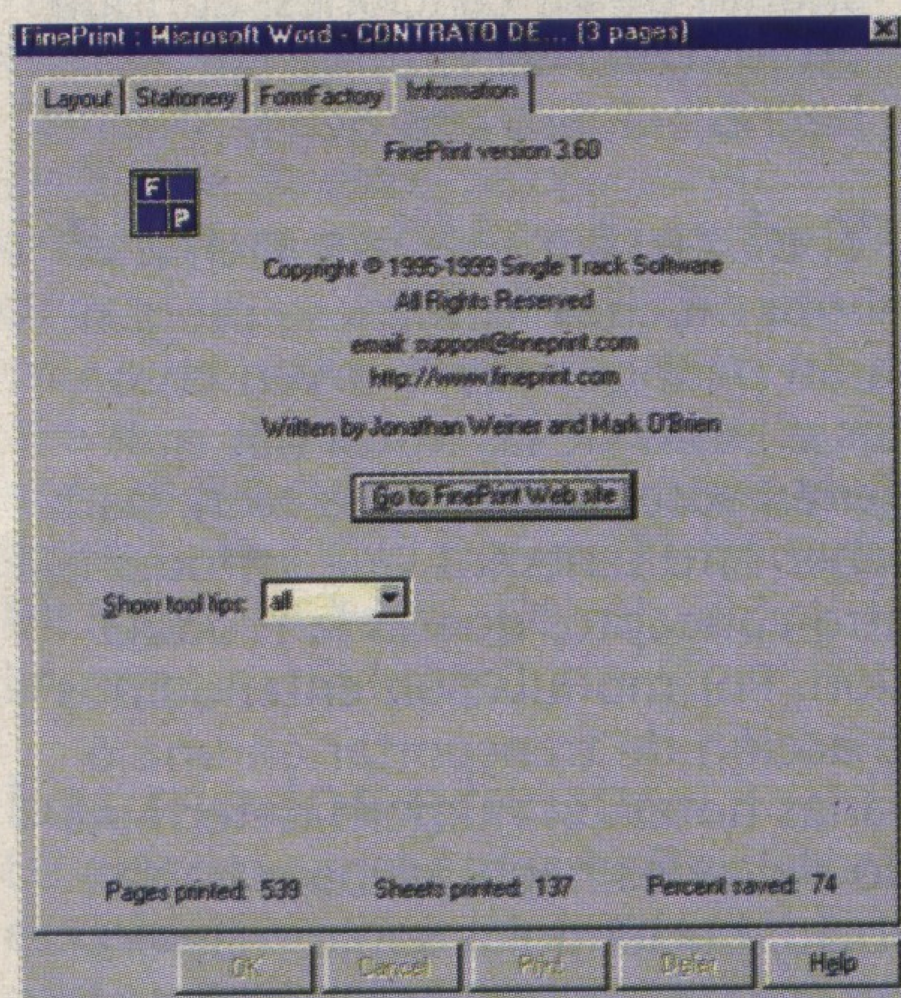
Instalación y configuración

La instalación del programa es muy sencilla, éste viene en un paquete de autoinstalación que busca versiones antiguas y las sobreescribe en caso necesario. No hace falta hacer absolutamente nada, ya que el programa se instala en el directorio Windows como un driver más de impresora.

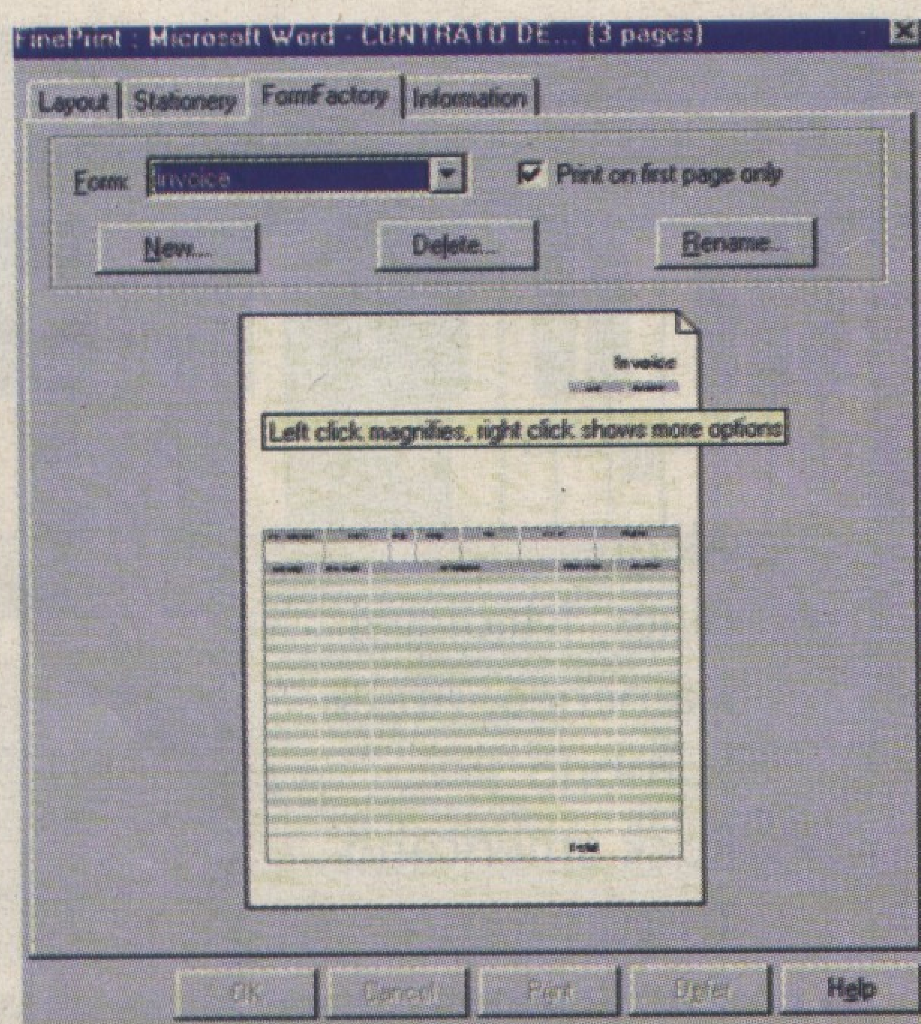
El funcionamiento teórico del programa es el siguiente: al instalarse como un driver de impresora los trabajos que van a ser imprimidos de los distintos programas son enviados a él, que los recoge, los trata (coge varias páginas y las mete en una) y los envía al auténtico driver de la impresora, que es el que los imprime.

La configuración del programa es bastante sencilla, en Inicio→Configuración→Impresoras veremos cómo nos ha aparecido un nuevo driver: "FjnePrint driver", en cuyas propiedades podemos ver más o menos las mismas características que las de cualquier impresora, a excepción de la solapa "setup", en la cual, además de elegir el tipo de papel que utilizaremos y su orientación, deberemos elegir si queremos que, cada vez que imprimamos se nos presente el menú de *FinePrint*. Hay tres posibilidades: "before spooling", "after spooling" y "not at all".

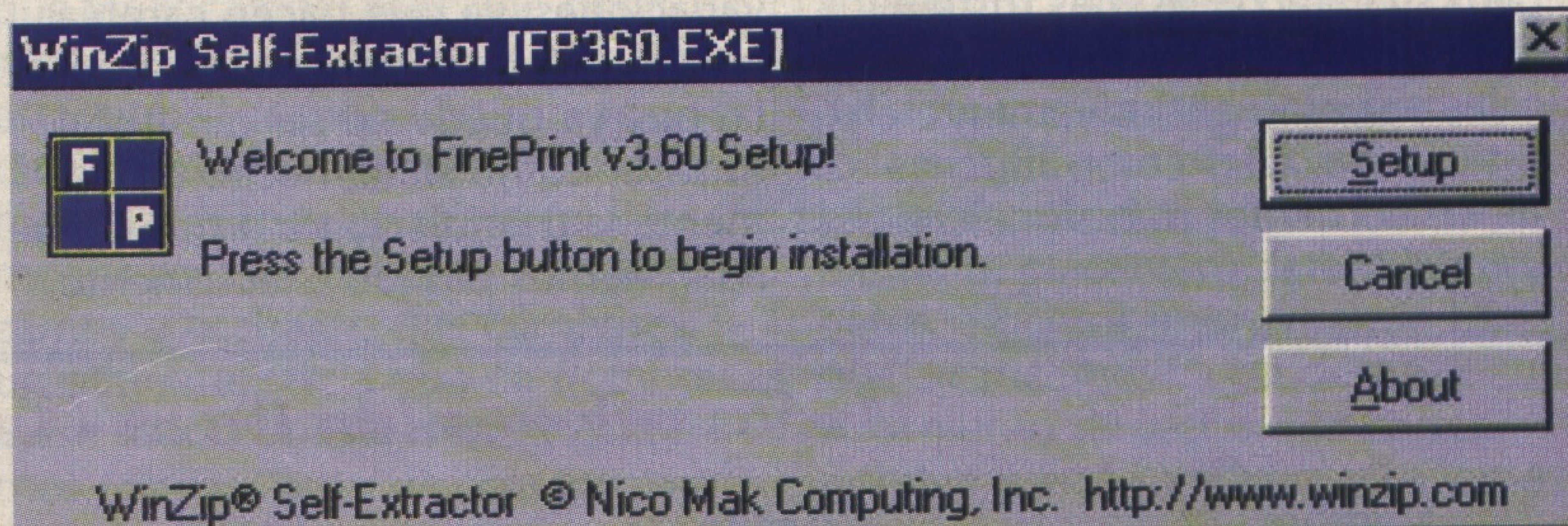
Before spooling: con esta opción los trabajos dirigidos a la impresora pasan casi automáticamente del programa al driver real de la impresora. Tiene como ventaja la rapidez en la que se empieza a imprimir el trabajo (muy bueno para documentos muy



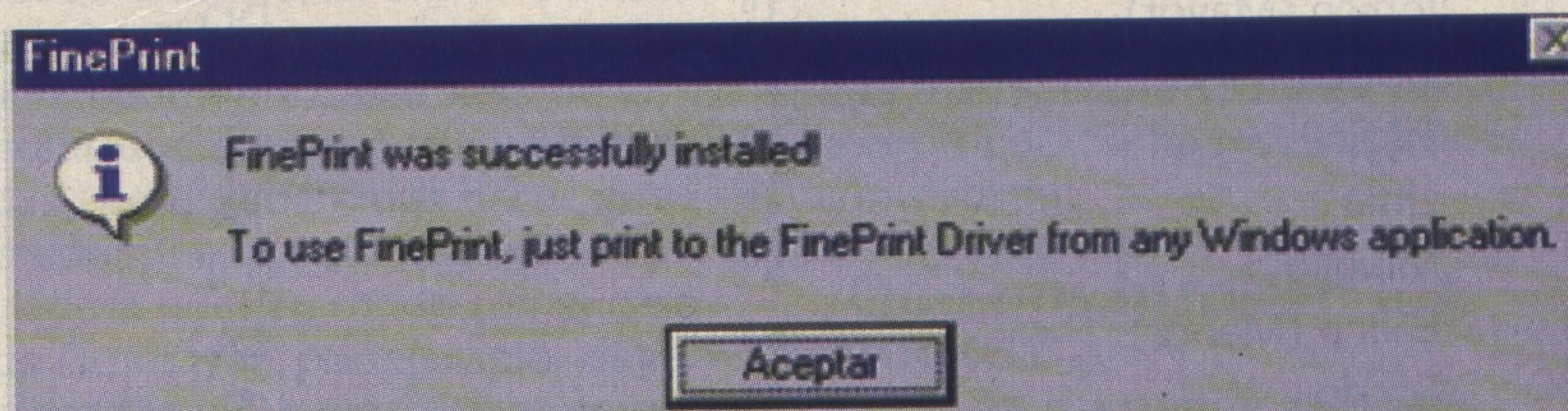
Información sobre el programa.



Así es el menú FormFactory.



Menú de instalación.



El programa se ha instalado satisfactoriamente.

grandes), y como inconveniente que no se puede ver el "preview" del trabajo que se va a imprimir.

After spooling: el driver de *FinePrint*, se espera a que todo el trabajo le haya sido mandado, entonces se muestra el diálogo de *FinePrint*, en el cual podremos ver el "preview" del trabajo en cuestión.

Not at all: con esta opción no se mostrará el menú de *FinePrint* cada vez que vayamos a imprimir algo, simplemente utilizará las opciones que hayan sido configuradas con anterioridad.

Un breve repaso

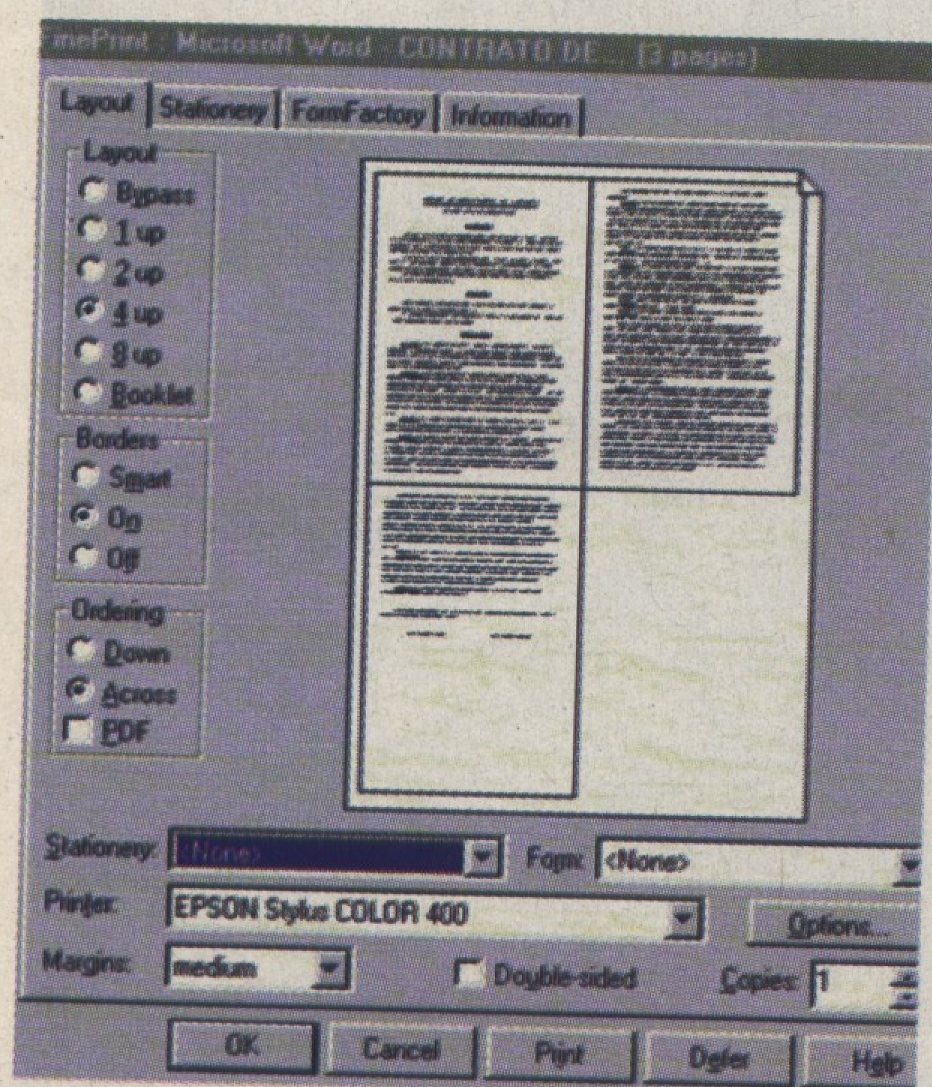
Como hemos dicho antes, en la solapa "setup" hay un botón, "advanced", que nos abrirá una ventana de configuración a la que podremos acceder también a la hora de imprimir, a no ser que hayamos puesto la opción "not at all".

Este nuevo menú tiene varias solapas que corresponden a distintos submenús: "layout", "stationery", "FormFactory" e "information".

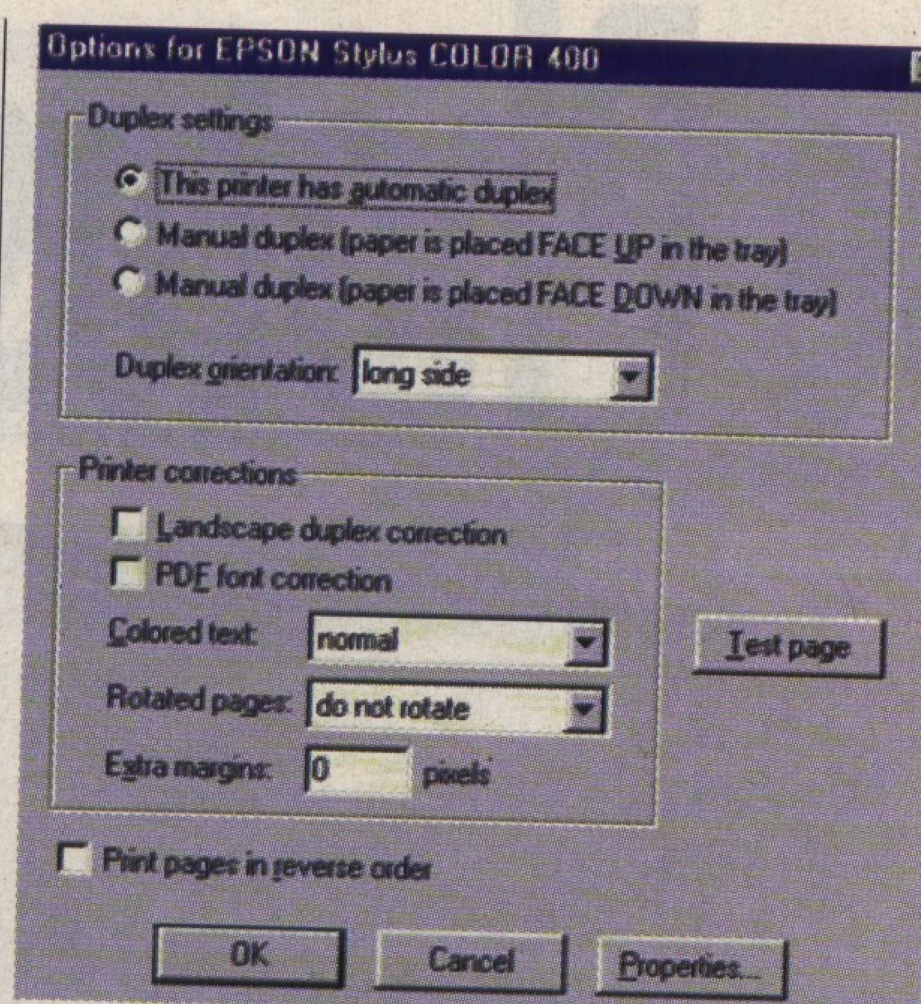
Solapa *Layout*. Aquí es donde configuraremos la forma de la página, si queremos que no reformatee el texto *(Bypass)*, o que imprima 1, 2, 4 u 8 páginas en una. También existe la opción "booklet", que sirve para hacer las dos páginas de un cuadernillo. Podremos poner bordes a las páginas o no, además de ordenarlas horizontal o verticalmente.

Elegiremos "stationery" y "form", además del tipo de bordes que le pondremos a la página, el número de copias y la impresora que vamos a utilizar para imprimir.

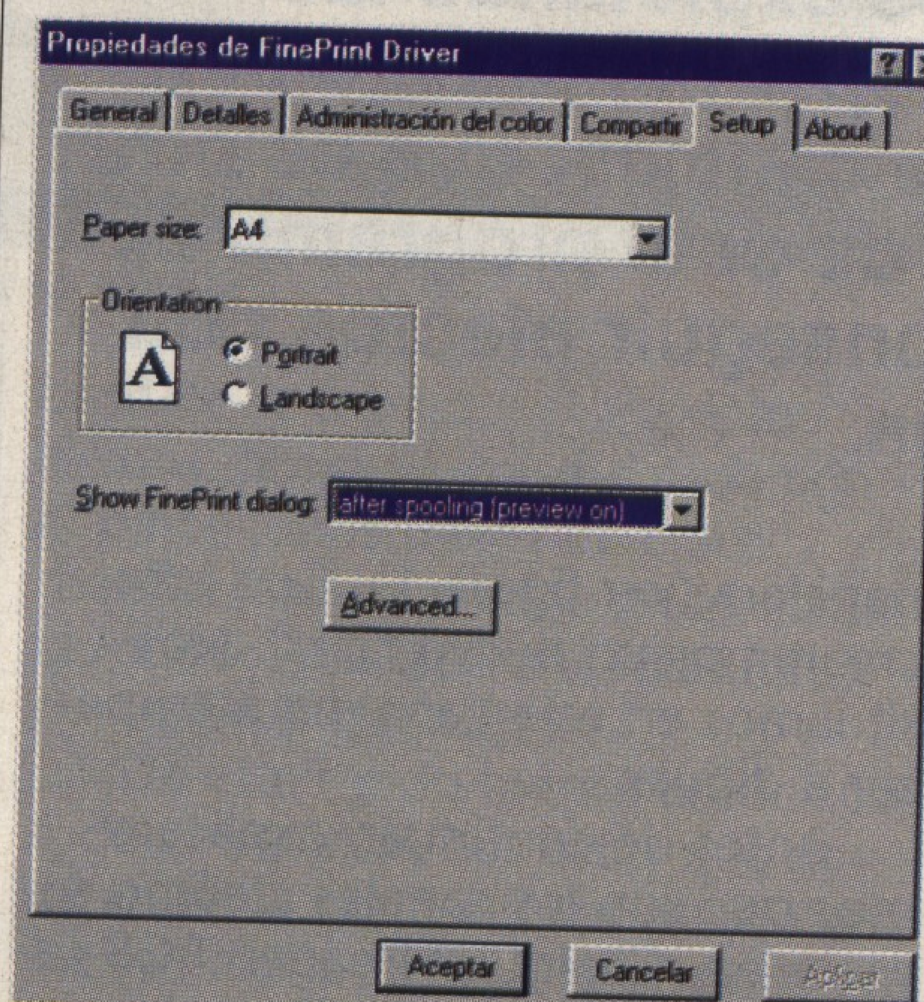
El botón "options" nos lleva a una nueva ventana donde le indicaremos al programa las opciones de la impresora que tenemos seleccionada. Lo que se denomina "duplex" es la opción de la impresora para que imprima por las dos caras de la misma hoja. En "automatic duplex"



Así es el menú Layout.



Aspecto del submenú Options de Layout.



Propiedades de FinePrint driver.

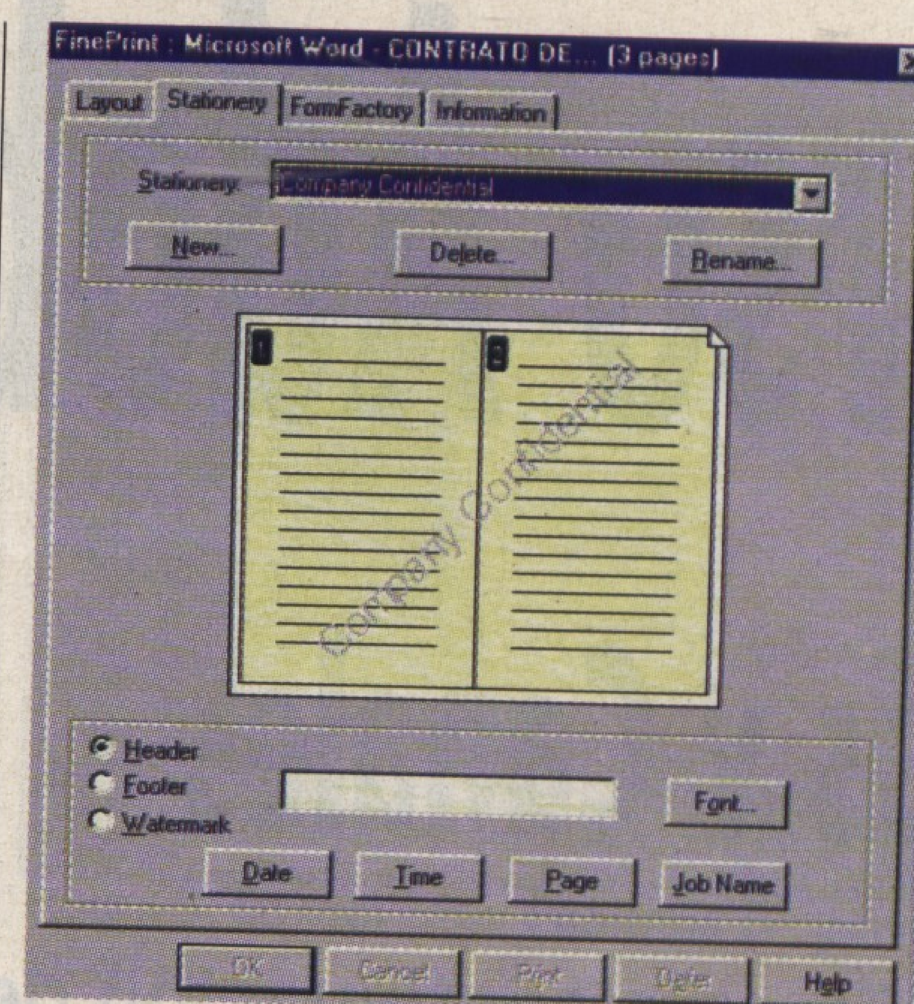
la impresora se encarga por si misma de imprimir por las dos caras del papel. Elegiremos la opción "manual duplex" si para imprimir por las dos caras tenemos que dar la vuelta a la hoja manualmente y poner la cara ya impresa visible.

Solapa *Stationery*. El programa nos da la opción de poner cabecera y pie a cada página que se imprima, además de lo que se llama "watermark", que no es más que un texto semitransparente que se escribe inclinado a todo a lo largo de la hoja, para indicar el tipo de documento que se está imprimiendo.

Se pueden grabar "stationeries" personalizados y en un momento posterior cargarlos para utilizarlos en otros documentos, facilitándonos así el formatéo de los mismos.

También se pueden incluir, tanto en la cabecera, como en el pie de página y el "watermark", la fecha (date), la hora (time), el número de página (page) y el nombre del trabajo (job name), o cualquier combinación de éstos y texto libre. Se pueden elegir el tipo de fuente y su tamaño, tanto para la cabecera como para el pie de página y el "watermark".

Solapa *FormFactory*. Esta utilidad sirve para incluir cualquier tipo de documento superpuesto al que



El menú Stationery.

vamos a imprimir. Esto sirve tanto para incluir membretes o logotipos en nuestros documentos (cartas, memorándums, o faxes) como para incluir el fondo de formularios, ya sean facturas, o cualquier otro tipo de documentos que sea necesario imprimirlos sobre un papel con un fondo determinado.

En este caso también es posible crear nuevos formularios y agregarlos para su utilización posterior. Es posible imprimir el logotipo o membrete únicamente en la primera página o en todas.

Para grandes reducciones hay que tener en cuenta la resolución de la impresora o fax

Solapa *Information*. En este menú se nos dan todos los datos del programa, además de un breve resumen sobre cuantas páginas hemos impreso y en cuántas hojas lo hemos hecho, además del porcentaje de ahorro según estos datos.

Para terminar

Ya hemos visto cómo puede ayudarnos este programa: aparte de ahorrarnos papel, nos da varias posibilidades, incluir membretes en los documentos, la fecha o la hora, fondo para formularios, o las "watermarks", que son verdaderamente útiles.

Hasta el próximo número de la revista. Espero que os haya sido útil.

Javier Fernández

Si queréis mas información sobre FinePrint, los datos de Single Track Software son:

E-Mail:	support@fineprint.com
Web:	http://www.fineprint.com
Correo:	Single Track Software 701 Minnesota St #201 San Francisco, CA 94107
Fax:	415-695-4081

Curso Adobe Photoshop (III)

Capas y Canales

En esta continuación del curso iniciado hace dos números continuamos explotando las posibilidades de *Photoshop* en el trabajo con capas, así como el uso de canales, que es otra herramienta muy potente de este programa de diseño, e intentaremos conseguir efectos con ambos a modo de práctica.

En nuestra tercera entrega del curso, a modo de ejemplo de capas y canales, haremos una interfaz para una página web o también para el menú principal de un videojuego. En el CD-Rom que acompaña a la revista encontraremos dos ficheros PSD, uno llamado "textura.psd" y el otro "interfaz.psd" que corresponde al ejemplo del artículo terminado. El diseño de la interfaz ha sido bastante simple. Deberíamos pensar un poco más la distribución de los elementos en él y "adornarlo" con más elementos.

Vamos a realizar una interfaz envejecida, como si hubiese sufrido el castigo de muchos clics de ratón

La forma de la interfaz

Abrir la textura que hemos utilizado para el ejemplo (textura.psd). Lo primero que tendremos que hacer es dibujar la forma que tomará nuestra interfaz. Para ello deberemos crear un canal nuevo. Pincharemos en la pestaña de canales (channels) y luego en el botón de la parte inferior de este submenú que está al lado de la papelera (ver figura 1). Con esto crearemos un nuevo canal. Nos situamos en este canal para empezar a dibujar.

Antes de continuar, explicaremos qué son los canales y cómo se manejan. Cualquier imagen RGB se compone de 3 canales (Rojo, Verde y Azul). Cada uno de estos canales está en escala de grises y guarda el porcentaje de ese color primario que compone la imagen RGB. El usuario puede crear sus propios canales para guardar información de máscaras y fragmentos de la imagen. Como era de esperar, los canales que defina el usuario también estarán en escala de grises.

En nuestro ejemplo hemos utilizado un canal para definir la forma que tendrá nuestra interfaz. Si vamos a usar un canal como máscara de selección debemos saber que los tonos claros (desde el blanco, pasando por grises claros...) son los que se van a seleccionar y los tonos oscuros (grises oscuros y negro) serán ignorados. Como queremos una selección limpia, la forma de nuestra interfaz se hará en blanco sobre fondo negro.

Seleccionamos el blanco como color de primer plano y con formas básicas (cuadrados, rectángulos, círculos y elipses) dibujamos la interfaz (ver figura 2). Nos ha quedado demasiada rectilínea. Suavizaremos los bordes con el siguiente truco: menú Filtro/Desenfocar/Desenfoque gaussiano (Filters/Blur/Gaussian Blur), dando un valor de 6 al radio aproximadamente. Acto seguido pinchamos en Imagen/Ajustar/Niveles (Image/Adjust/Levels) y situamos los valores como muestra la figura 3. El resultado obtenido es una interfaz con bordes suavizados. Usamos este canal como máscara de selección. Seguidamente pinchamos sobre la imagen en pequeño del canal y arrastramos hasta el primer botón inferior empezando por la izquierda (una pequeña circunferencia punteada). Veréis que la forma que habíamos dibujado

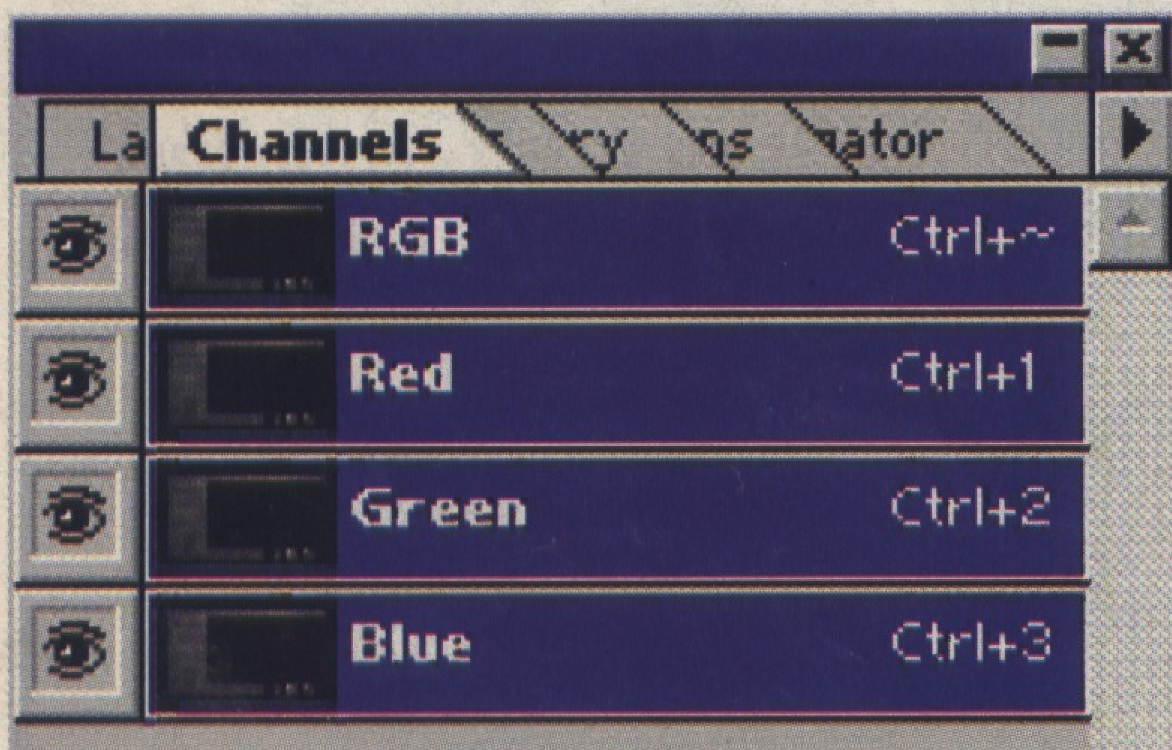


Figura 1. El menú Canales.

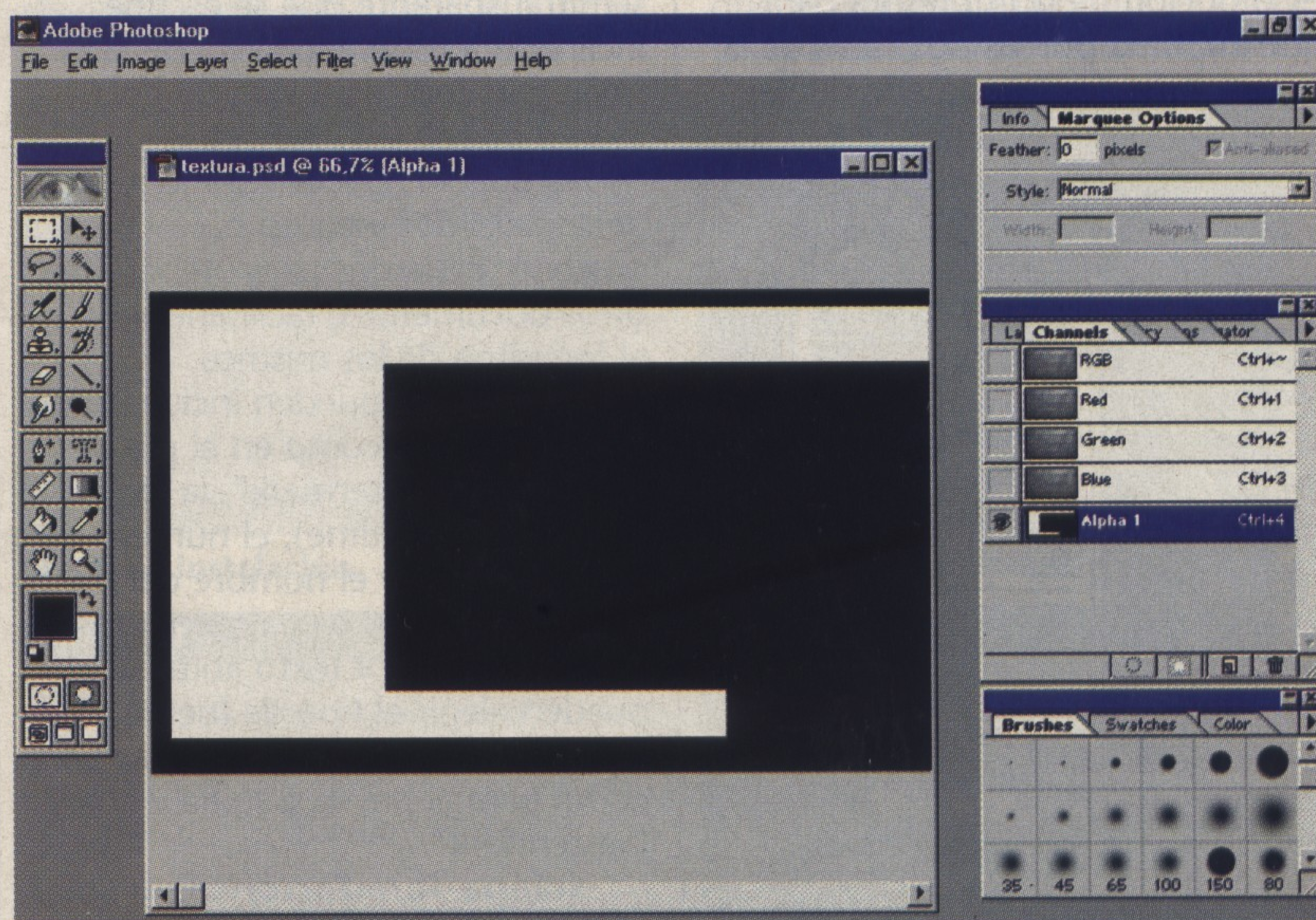
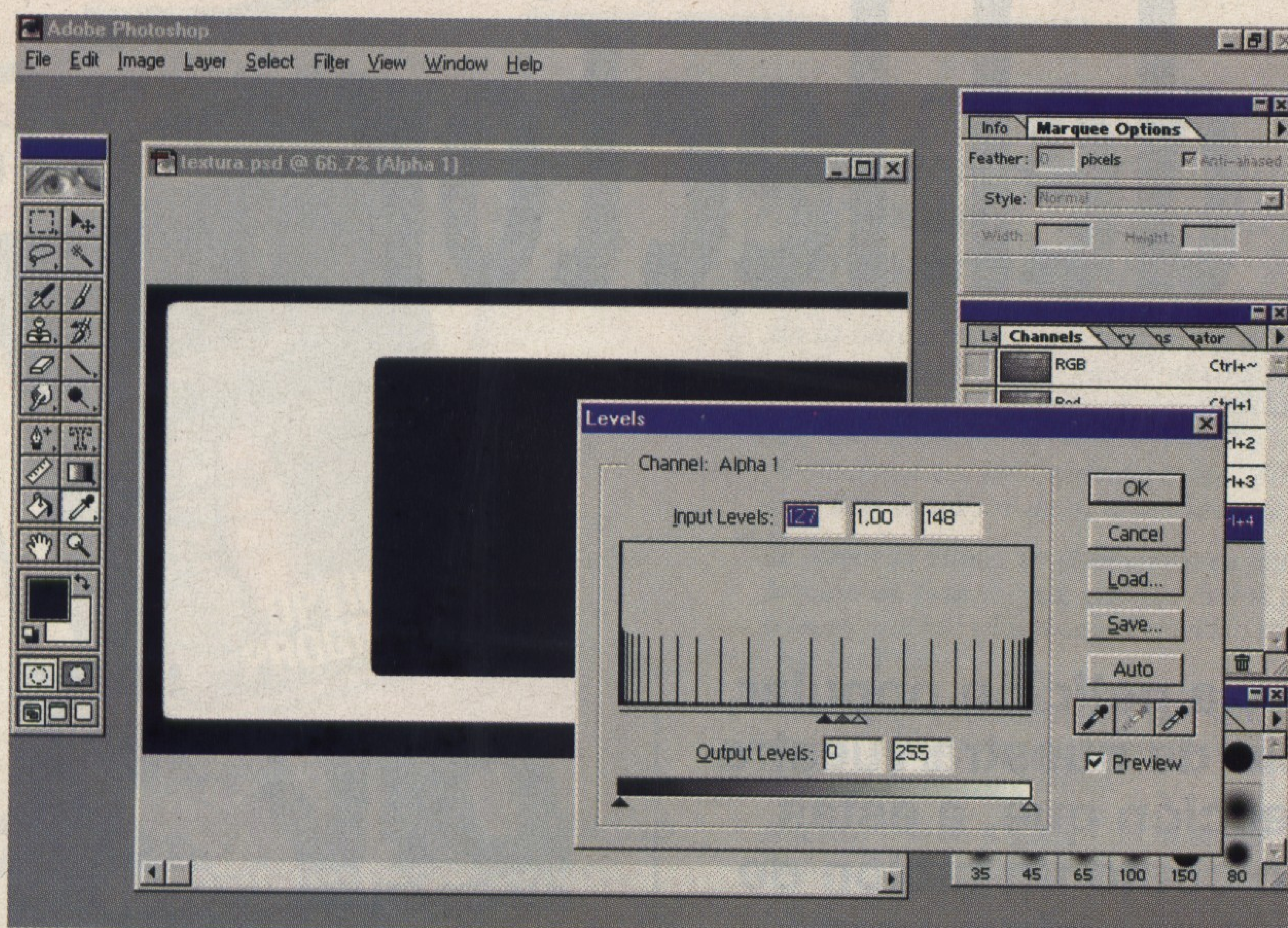
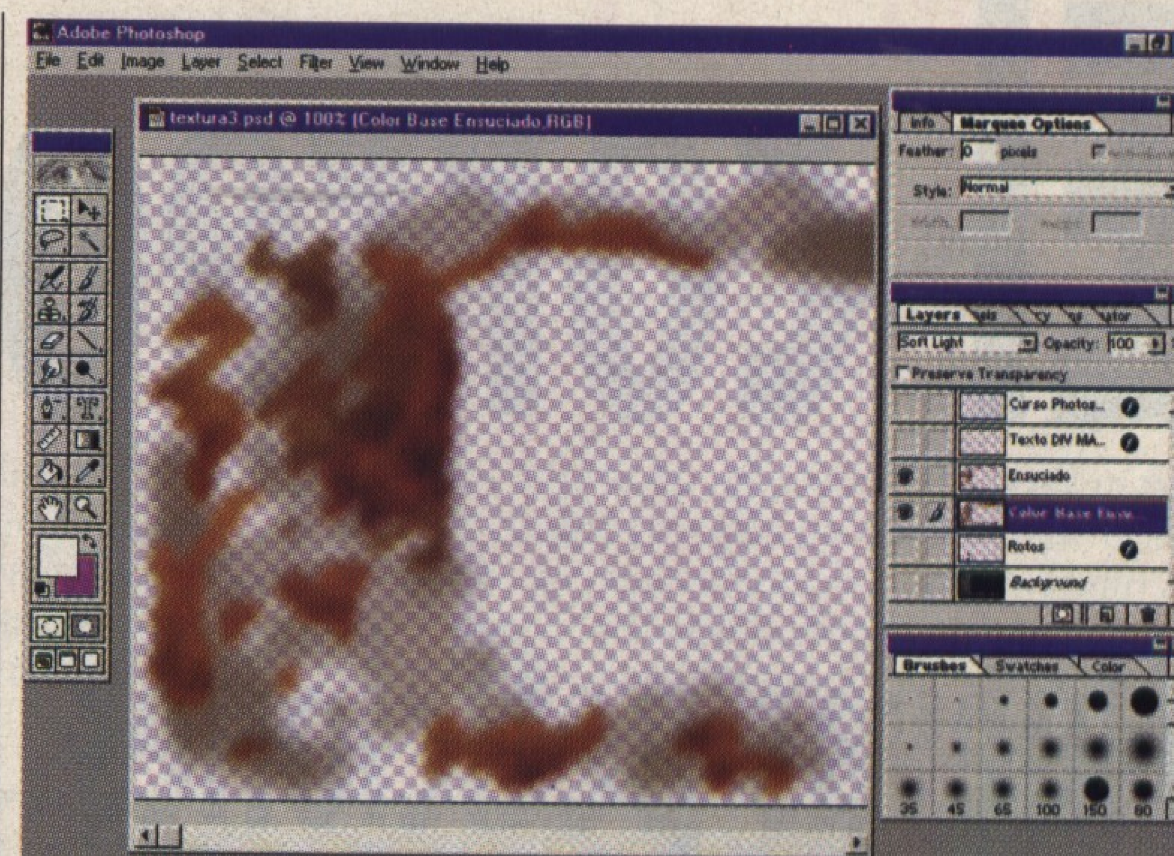


Figura 2. La interfaz dibujada con rectángulos y círculos.


Figura 3. Ajustando los niveles para quitar la "niebla".


Figura 5. En esta capa está toda la "porquería" de la interfaz.

queda seleccionada. Ahora pinchamos en el canal RGB (que será el primero), menú Selección/Invertir (Select/Inverse) y borramos esta parte (figura 4). La forma de nuestra interfaz está terminada.

Ensuciando el material

Vamos a envejecer un poco nuestra interfaz, dándole apariencia de haber recibido muchos "clicks" de ratón. Tanto desgaste tiene que mostrarse con suciedad, magulladuras.

A partir de ahora trabajaremos con capas. En el número anterior de Divmanía trabajamos con capas pero siempre en modo normal. Si desplegamos el submenú de modos de trabajo con capas, pinchando en el menú desplegable de la pestaña "capas" (Layers), veremos que hay múltiples opciones para seleccionar. Todas trabajan con el modo de fusionar el color base de la capa. En el ejemplo hemos dibujado en una capa con el pincel unos trazos de color marrón y se ha puesto en esa capa el modo "Luz Suave" (Soft Light) de fusión (figura 5).

De este modo hemos ensuciado todo el material. Los rotos que aparecen en la parte inferior de la interfaz se han hecho con la herramienta de selección "Lazo". Hacemos una nueva capa y dibujaremos con el ratón la forma que va a tener el "arañazo", y la rellenamos de color blanco. Movemos la selección un pixel abajo a la izquierda. Rellenamos de color negro. Menú capas/efectos/sombra interior (Layers/effects/Inner Shadow). El modo de la capa será "Luz Intensa" (Hard Light). Para dar sensación de chapa, los bordes estarán levantados. Daremos brillo y sombras a la chapa como explicamos en la primera lección del curso.

En el texto de la esquina superior derecha (Curso de Photoshop-Lección 3) se usó el efecto "Inglete" y "Relieve" (Bevel and Emboss), que permite añadir sombres y luces que crean un efecto de volumen saliente.

Si quisiéramos utilizar nuestra interfaz en la Red, habría que recortar la imagen en 3 partes para su adaptación al HTML. Podéis ver el resultado de recortarla y optimizarla (se guardó finalmente en formato JPG) en la figura 6.

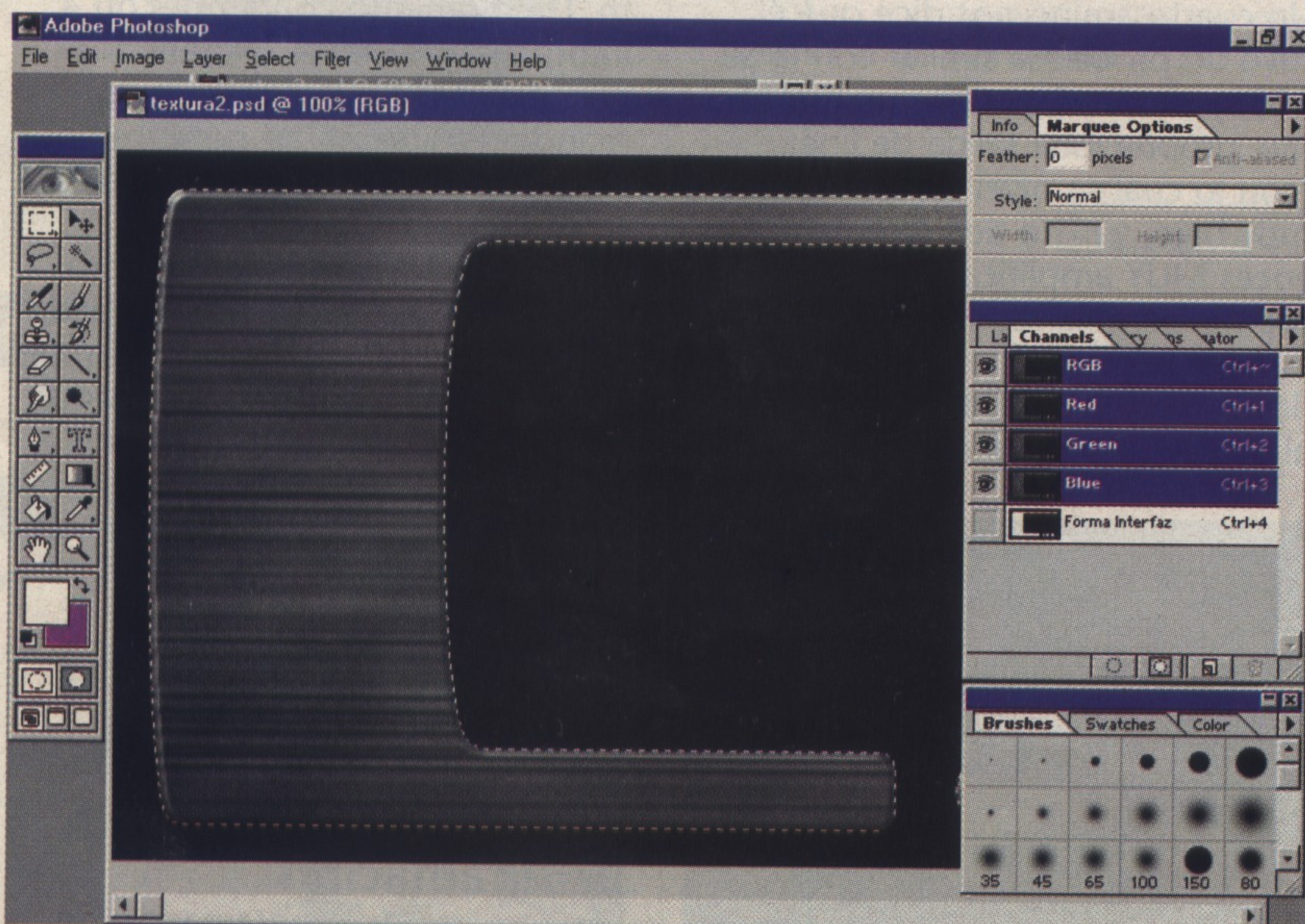
> Cualquier imagen RGB va a estar compuesta de tres canales: rojo verde y azul

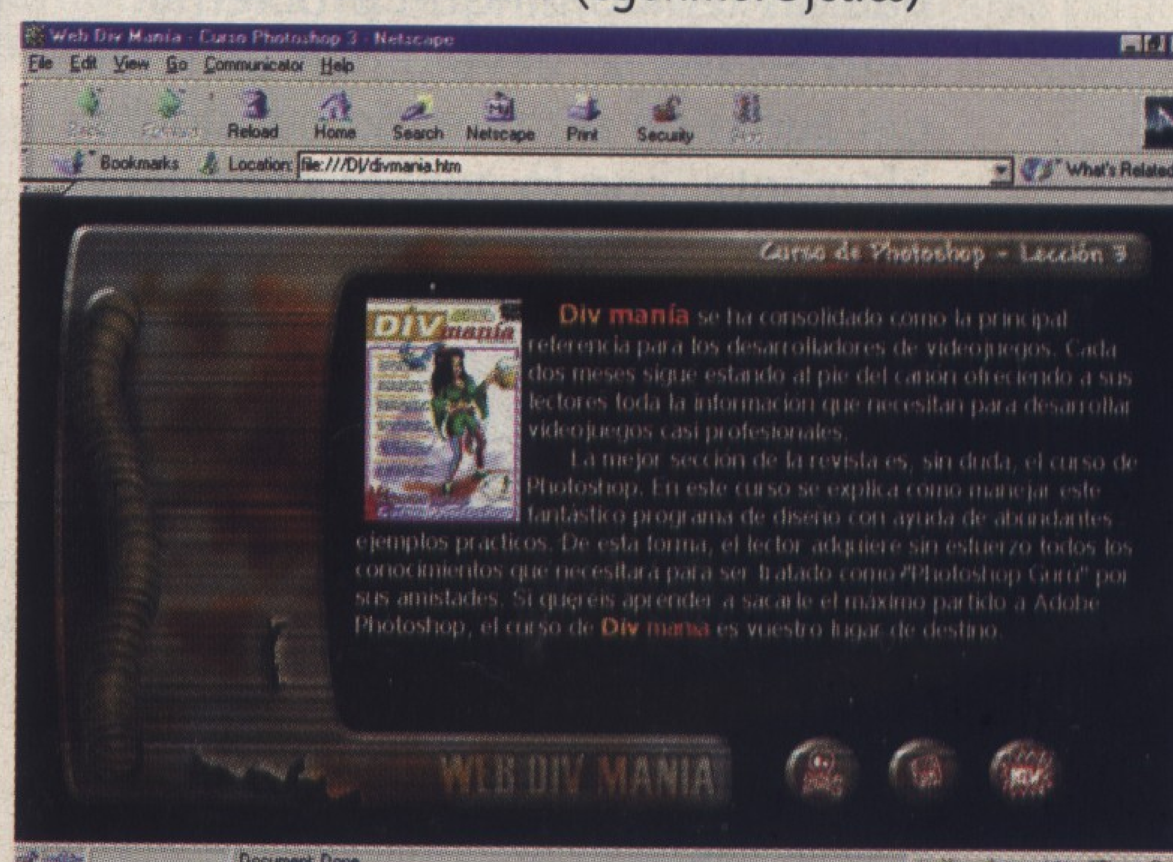
El ejemplo terminado está en el CD-Rom que acompaña a la revista. Abrir la imagen «interfaz.psd» y cambiar el orden de las capas, modificar los modos de fusión... para familiarizarse con los efectos que podemos conseguir con capas. Sería bueno que hicierais vuestra propia interfaz partiendo únicamente de la textura que hemos facilitado. Para cualquier consulta referente a este artículo, el buzón de correo electrónico de la redacción queda abierto en cgonmor@jet.es.

Dentro de dos meses...

Continuaremos con el empleo de capas y canales. Daremos tablas resumen para usar más rápidamente y con precisión estas herramientas de *Photoshop*. Hasta entonces, un saludo.

Carlos Glez. Morcillo (cgonmor@jet.es)

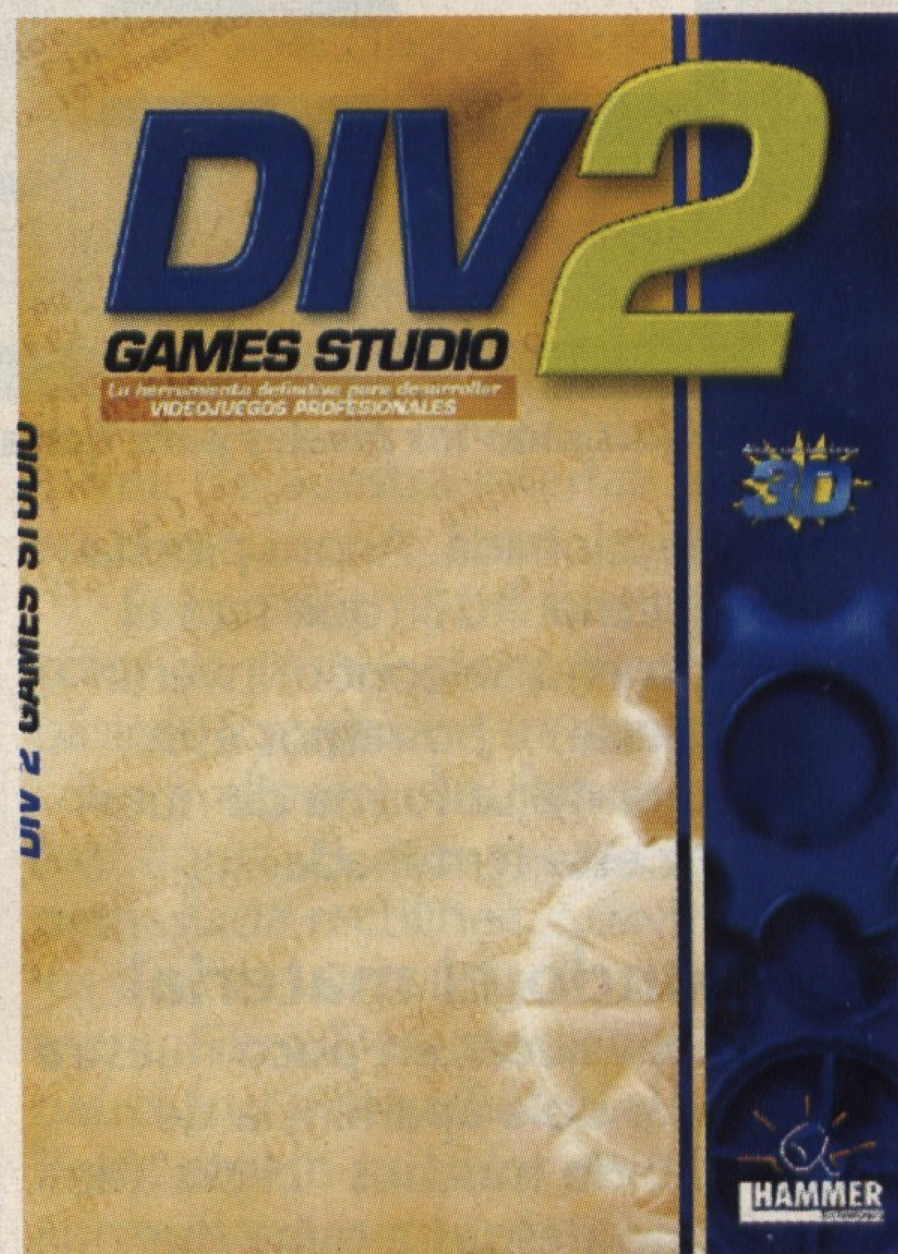

Figura 4. Recortando lo que no nos vale.


Figura 6. Navegando en nuestra interfaz.

El correo del lector

Nuevas peticiones

Como podéis comprobar, y a petición popular, hemos ampliado el espacio de nuestra/vuestra sección de correo. Una sección que, a estas alturas, se ha consolidado como un tablón de anuncios. Esta vez, además nos sirve para contestar algunas dudas de nuestros lectores. Allá vamos.



Resulta que cuando me compré el DIV 1 todo iba bien, excepto la limitación de formatos de sonido, ya que yo, a lo que más importancia concedo es a la música y el sonido del juego, así que en cuanto me entero de las novedades del DIV 2 me lo compré todo ilusionado. Pero el problema es que a pesar de haber configurado correctamente la tarjeta de sonido, y ajustado al máximo los niveles de audio, tanto del Windows como de DIV 2 el sonido se escucha increíblemente bajo, lo cual es un gran problema para mí. ¿Es normal esto? Si no ¿qué pasa, qué debo hacer? Gracias.
Ivan Pérez Brea.

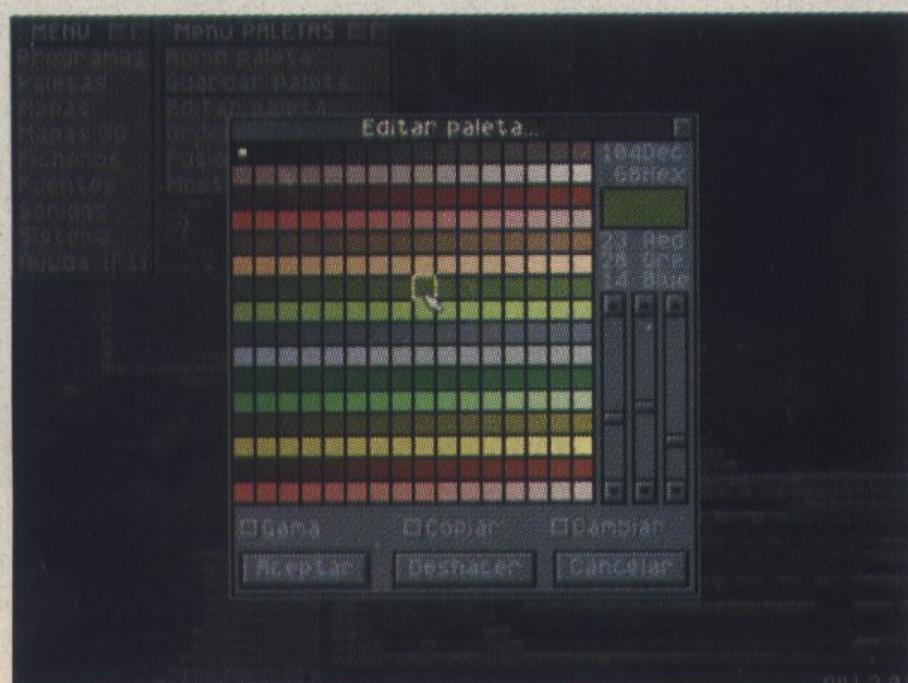
Pues parece que la cosa no es fácil, hemos sondeado entre nuestros colaboradores y nos han dado varias posibles respuestas. A continuación las reproducimos esperando que te solucionen el problema. Realmente no es ningún problema de tu tarjeta de audio. Si habías trabajado con DIV 1, seguramente no habrías tenido ningún problema. Pero en DIV 2, el volumen general del audio ha disminuido considerablemente, con lo cual es completamente normal que el sonido se escuche realmente muy bajo. La única solución posible a este problema es grabar tus archivos de audio a un volumen mucho más alto de lo normal con lo cual se te oirá bien en tus programas.

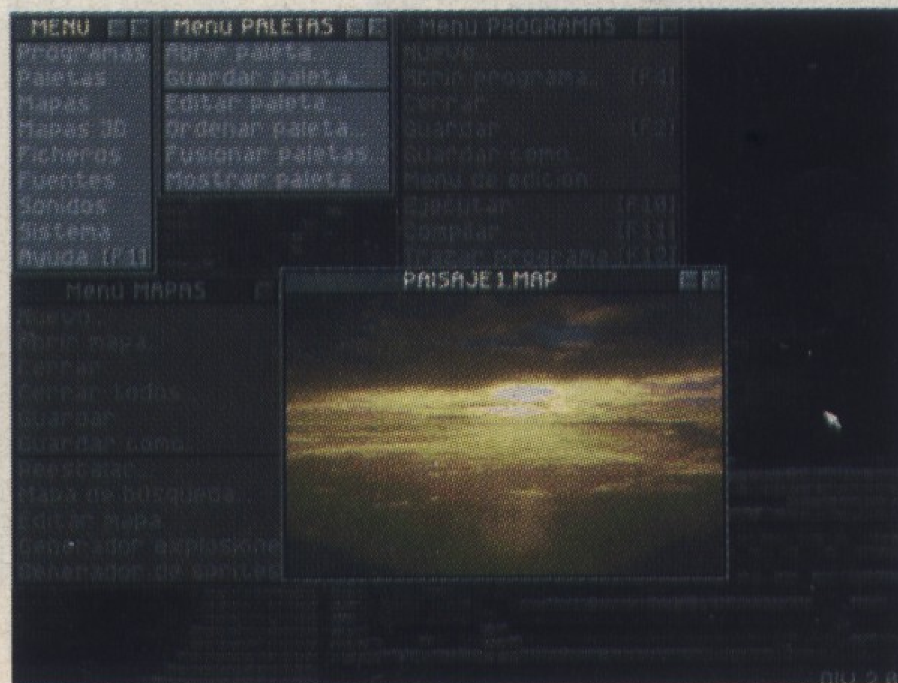
Más soluciones:

- Que pruebe subiendo el volumen del wav con el editor de sonidos.
- Que suba el volumen desde Windows (cosa que ya habrá probado).
- Que ajuste el volumen con los drivers para MS-DOS de la tarjeta, en caso de que tenga.

Hay otro Divero que dice que también le pasa, pero con subir el volumen de los altavoces con la ruedecilla le vale, aunque luego tiene que volver a bajarlo para que no suene todo lo demás muy alto.

Un cuarto amigo nos dice que él tuvo un problema similar con la SB 128, y es que ésta tiene un jumper, que proporciona salida de línea, (con lo cual su nivel de salida es muy bajo, y si no tienes unos altavoces MUY amplificados... pues eso...). Cambiando la posición del jumper se proporciona salida amplificada, con lo cual la mayoría de altavoces suenan alto y claro, (y sin necesidad de amplificarlos). Así que igual ésa es la pega... (si es que tiene TODOS los volúmenes al máximo, entradas, salidas, etc.).





Se buscan dibujantes

Solo quería saludaros, felicitaros por la revista, y comentaros que tengo un grupo de desarrollo. Estamos metidos con un juego de estrategia por turnos que creo, será muy bonito. De momento contamos con 4 infografistas, 2 músicos muy buenos, y al menos somos 6 programando. Posiblemente necesitemos algún dibujante, así que si alguien se anima...

Estamos apretándonos el cerebro, no utilizaremos DIV, prefiero trabajar con C++ y DirectX. Si alguien se anima puede mandarme muestras de su trabajo, prometo al menos contestar a todos. Requisitos imprescindibles: menos de 1 mega (no bajo más, nunca) y vivir en Madrid.

Bye, bye, gracias por vuestro tiempo, y suerte con la revista.
ospoley@santandersupernet.com

Curso de programación

Vendo curso de programación de videojuegos, en PASCAL. Muy com-

PROGRAMAR JUEGOS ES COSA DE NIÑOS SI TE SUSCRIBES

a Div Manía



Si deseas estar en la vanguardia del mundo de la informática, suscribirse a DIV MANÍA es un primer paso acertado porque...

- *Es la única revista escrita por y para los programadores de videojuegos. Nuestra redacción está compuesta por veteranos desarrolladores, expertos del entorno DIV, grafistas y muchos otros profesionales del software de entretenimiento que dan lo mejor de sí a los lectores.*
- *Te ofrece lo último en el delicado campo de la programación de videojuegos, con los títulos que se encuentran en proceso y los productos recién salidos del horno o de los PCs.*
- *Para seguir avanzando hay que saber echar la vista atrás y a la vez no olvidarse del futuro, y **Div Manía** empieza un nuevo camino.*
- *Nuestra revista presenta un look muy cercano a sus lectores, salido de las inquietudes de todos vosotros.*
- *Nunca nadie te ha ofrecido tanto por tan poco; nunca has tenido tan cerca la oportunidad de estar al día de lo último en programación por el mínimo esfuerzo de acercarte al quiosco o enviar nuestro cupón y recibir la revista en casa puntualmente cada dos meses.*
- *En el interior del CD-Rom encontrarás los elementos con los que todos los programadores sueñan.*
- *Somos como tú y conocemos, más o menos, qué se esconde dentro de tu cabeza. Y si no lo conocemos aún, lo aprenderemos gracias a ti.*
- *Ofrecemos las más diversas sorpresas, las más interesantes ofertas, para que no te olvides de que la programación siempre está viva.*

Además, el suscriptor tiene derecho a la siguiente oferta:

- **Con un año de suscripción (seis números) regalamos un producto a elegir entre:**
  "Programación Gráfica para PC"
  "Cómo programar en Ensamblador"
- **Con dos años de suscripción (doce números) regalamos DIV 2**



PROGRAMACIÓN GRÁFICA PARA PC
Libro programación

CÓMO PROGRAMAR EN ENSAMBLADOR
Libro programación

DIV II
Creación de juegos

Prensa Técnica se reserva el derecho de cambiar cualquiera de los regalos por otros de igual valor, en caso de agotar existencias.

Solicite su ejemplar enviando este cupón por correo, por fax: 91 304.17.97 o llamando al teléfono 91 304.06.22 de 9:00 a 19:00 h.

CUPÓN DE SUSCRIPCIÓN ANUAL A DIV MANÍA

Deseo suscribirme a la revista **DIV MANÍA** acogiéndome a la siguiente modalidad:

- ☐ Suscripción: 1 año (6 números) por sólo 5.970 ptas. ☐ Correo certificado 1 año: 1.500 ptas. adicionales.
- ☐ Suscripción: 2 año (12 números) por sólo 11.940 ptas. ☐ Correo certificado 2 años: 3.000 ptas. adicionales

Desde el número ____

Además recibiré gratis:

- ☐ Por 1 año de suscripción: uno de los siguientes productos:
- ☐ Por 2 años de suscripción: **DIV II**

☐ Programación Gráfica para PC ☐ Cómo programar en Ensamblador

Nombre y apellidos ....
Domicilio ....
Población ....
C.P. .... Provincia ....
Telf. .... Profesión ....
DNI/NIF: ....

FORMA DE PAGO:

☐ Con cargo a mi tarjeta VISA nº ________ más gastos de envío
Fecha de caducidad de la tarjeta ....
Nombre del titular, si es distinto ....

☐ Domiciliación bancaria, más gastos de envío
Población ....
Ruego a Vd. que se sirva cargar en mi:

☐ cuenta corriente
☐ libreta de ahorro número

ENTIDAD	OFICINA	DC	Nº CUENTA

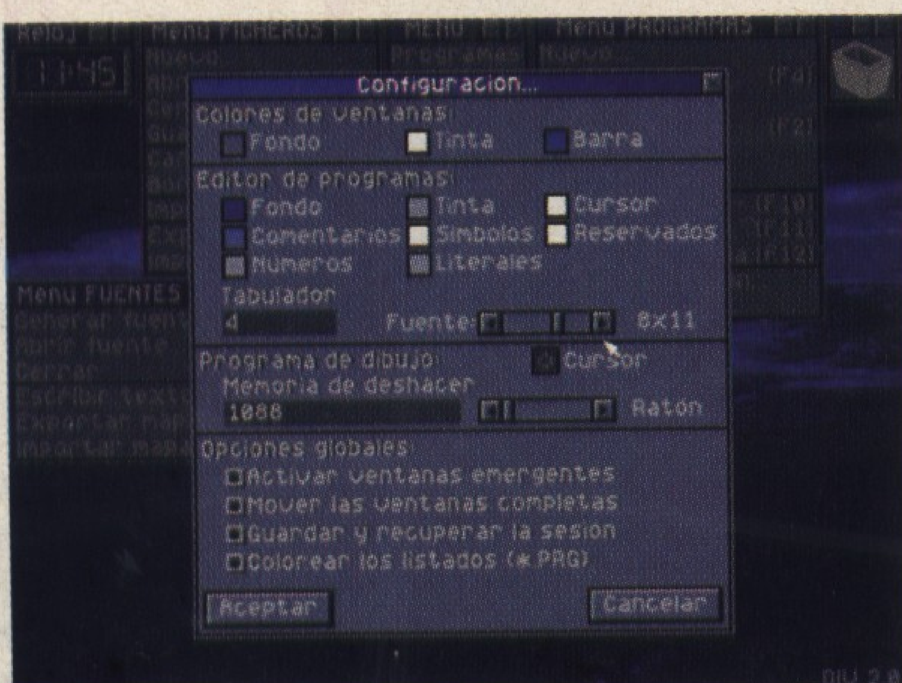
el recibo que será presentado por PRENSA TÉCNICA S.L. como pago de mi suscripción a la revista **DIV MANÍA** más gastos de envío.

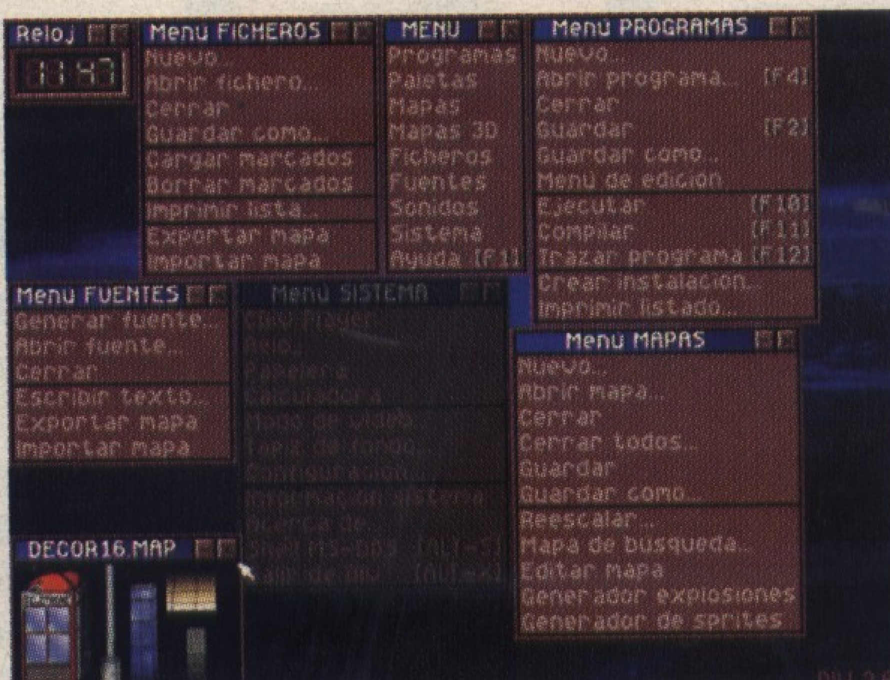
FIRMA:

☐ Contrarreembolso del importe más gastos de envío.
☐ Cheque a nombre de PRENSA TÉCNICA, que adjunto más gastos de envío.
☐ Giro Postal (adjunto fotocopia del resguardo) más gastos de envío.

Prens@ Técnic@ de libros y publicaciones

C/ Alfonso Gómez, nº 42 Nave 1-1-2. 28037 Madrid. Tfno: 91 304 06 22. Fax: 91 304 17 97
e-mail: maspc@prensatecnica.com. http://www.prensatecnica.com







pleto, multitud de accesorios y utilidades. Pide más información escribiendo al apartado nº 70 - 08830 San Boi (Barcelona). Oscar Gaya.

Busco programador y grafista

Quiero formar un grupo de programación de div, necesito programador y grafista que trabaje en 3D Studio max, a ser posible de mi población, que es Motril (Granada), o alrededores para hacer un juego que decidamos nosotros, me da igual la edad.
Nick: Lanthi
Kie@retemail.es
Lanthi@ozu.es

Urgente

Grupo de desarrollo precisa de los servicios del mayor número posible de grafistas 2D para realizar el mejor "shooter" estilo "Operation Wolf". Se tratará de un videojuego serio con personajes y escenarios digitalizados. También nos vendría bien contar con algún programador en DIV2 con conocimientos avanzados y con gente que pueda aportar sonidos y/o músicas S3M o XM. Escribid rápido, ya que somos estudiantes de informática y dentro de poco empezamos fuerte, así que nos podemos echar atrás si no contamos con gente suficiente.

```
  ("`-"-/").___..—''''`-._
   `6_ 6 )   `-.  (     ).`-.__.`)
   (_Y_.)'  ._   )  `._ `. ``-..-'
    _..`—'_..-_/  /—'_.' ,'
 (((),-"  (((),'  (((.-´
```

Luis Morante Fernández
luismorante@jet.es

Programador con experiencia

Se ofrece programador de Div Games Studio, para realizar proyectos en conjunto con otras personas, experiencia con otros lenguajes (Delphi, C++, Basic.....), así como en realización de gráficos 3D, ya he realizado unos cuantos programas y me gustaría seguir desarrollando muchos más con ayuda de otras personas.
Daniel García Alonso.
Dagal@Alehop.com

Para komenzar

Hola me llamo Carlos Hileno y me compre el div2 hace unos meses, nunca me avia interesado por la programacion de juegos porque lo beia una cosa muy dificil pero un buen dia en la revista micromania (seguro ke saveis cual es)vi un anuncio del div2 i me parecio interesante. Aquella noche soñe ke hacia una aventura grafica tipo "the day of the tentacle" pero + kachonda. Me gustaria konocer a usuarios de DIV ke vivieran cerca de Cerdañola para hablar i para komenzar a hacer algun proyecto tipo "day of the tentacle".
hileno@mixmail.com

Grafistas para juegos de rol

Busco Grafistas 2-D, 3-D y algún programador para realizar conjuntamente un buen juego de Rol en 2D tipo *Zelda* o en 3-D tipo *Lands of lore*. Interesados enviar e-mail a: Yago.
YTK@alehop.com y *YagoTK@teleline.es*

Byotech

Necesitamos grafistas 2D: Que sepan dibujar con estética japonesa y que dominen técnicas de tratamiento de gráficos predibujados. Programadores: Que sepan programar juegos bajo Windows. Es decir, que dominen Visual C++ y las Direct X. La colaboración en nuestro proyecto tendrá que ser voluntaria y con la única remuneración a repartir de los premios ganados en concursos. Importante que los colaboradores residan en el área de BARCELONA (somos de Sta Coloma y Badalona), para facilitar las reuniones, y las entregas de material. Interesados contactar con el Webmaster:
roberto.bar@mx3.redestb.es

Crystal Wave Studios

ATENCION! Crystal Wave busca grafistas 2D, artistas y músicos. Si estas interesado escribe a:
pagina.de/CrystalWave

Cukker Games

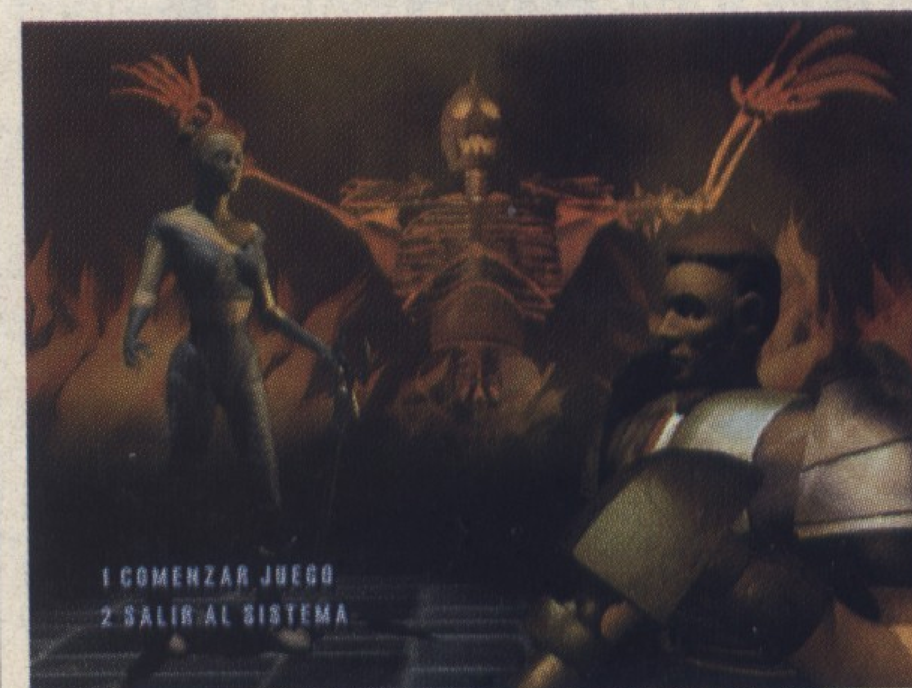
Pagina dedicada al div que tiene como principal objetivo enseñar en div. El grupo necesita gente que se quiera unir a su ellos. Interesados escribir a
C_urro@hotmail.com

D&S

Por lo pronto somos tres personas con diversos colaboradores casuales. El programador y organizador de todo este rollo soy yo, David Contreras Sánchez-Matamoros (D), Salvador Martín Naval es el director gráfico, grafista y encargado de la música (S) y el tercero es Iván Antúnez Barbancho, un grafista de primera que se quiere dedicar a la infografía. Si queréis uniros o simplemente colaborar escribidnos a
davidco@arrakis.es

Grupo –DIV Games Bilbo–

Ya somos unos cuantos en el grupo y hemos empezado un desarrollo, pero todos los que estéis interesados mandad un e-mail con la referencia Grupo-Div a la dirección de correo, indicando que tipo de tarea podéis realizar: gráficos, sonido, programación, etc... Entre todos afrontaremos los distintos proyectos u otros que fueran posteriormente propuestos y elegidos por el grupo. Sería muy interesante que varios grafistas, muchos grafistas, se interesaran, porque desde luego es lo que más escasea, aunque también serán necesarios varios programadores. Recuerdo que el tema del audio también esta ahí y es importante (creo que ya es posible la utilización de los ficheros mod). En cuanto a las producciones que logremos llevar a cabo será el grupo quien decida que se hace con ellas freeware, shareware, otros ...
gcuadrado@netflow.es
patxisanchez@jet.es



Contenido CD-Rom



Un número más acercándonos a los lectores y tratando de facilitarles un poco la vida. Como en todos los números hemos seleccionado las utilidades que consideramos imprescindibles y más útiles para vosotros y os la ofrecemos en el CD de la revista, además de demos, ejemplos de nuestros cursos, y una revista entera.

DEMOS

Ancient Evil

Un juego de rol medieval en el que tendremos que luchar contra numerosos enemigos para desvelar el misterio de La Cripta de los Antiguos. Numerosas armas y hechizos, objetos mágicos, tesoros y un desafío...

Winter Sports

Toda la emoción de los juegos de invierno es lo que nos ofrece este nuevo producto de Hammer Technologies. En esta segunda parte del fabuloso *Snow Wave* tendremos la oportunidad de practicar 3 deportes de invierno apasionantes, el esquí, el snow, y el snowmobiling, o carreras de motos de nieve.

Seven Years War

Un curioso juego de estrategia con un planteamiento original. Como la mayoría de los juegos del género se sitúa en la edad media; sin embargo, la situación de la aventura es otra: la acción se traslada a las lejanas tierras de Oriente. Los personajes de la aventura son los habituales, sin embargo, lógicamente, están adaptados al nuevo contexto. Ahora debemos hacernos cargo de campesinos asiáticos, guerreros mongoles y samurais japoneses, entre otros. En definitiva un buen programa para los amantes de los juegos de estrategia que sin duda encontrarán la manera de disfrutar con las aventuras que propone.

Div Games Studio 2

Ya lo sé, otra vez, sí, pero creedme, seguimos recibiendo en la redacción muchas cartas y emilios pidiéndonos una demo de la nueva entrega del DIV. Todos los números pensamos que va a ser el último que tengamos que incluirla pero las peticiones terminan por convencernos de lo contrario. Además es gratis y a nadie le estorba. Por otra parte, los que se acerquen por primera vez a nuestra revista es muy posible que no hallan tenido oportunidad de ver las maravillas del *DIV 2* y esta es una buena oportunidad para ello.

EXCLUSIVA

Revista Divnet

Para todos aquellos que no estén enganchados a la red pero tienen curiosidad por saber que es lo que se cuece en esta revista electrónica la hemos incluido en el CD. Tal vez de esta forma os convenceremos para dar el paso y conectaros por fin a Internet.

JUEGOS GANADORES

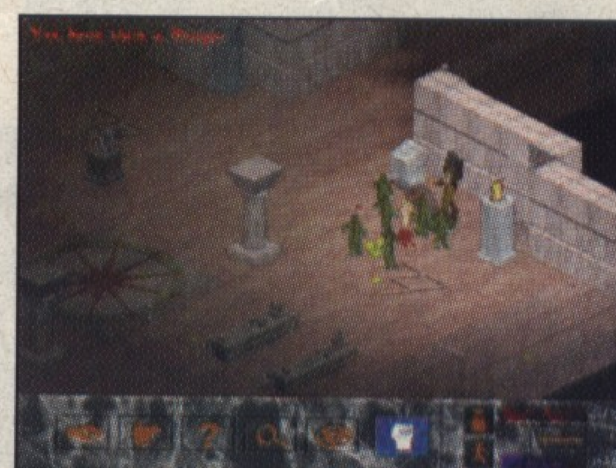
Aselo

Un matamarcianos al más puro estilo de las recreativas. Un viaje al pasado, cuando aquellas sencillas máquinas nos atraían tanto.

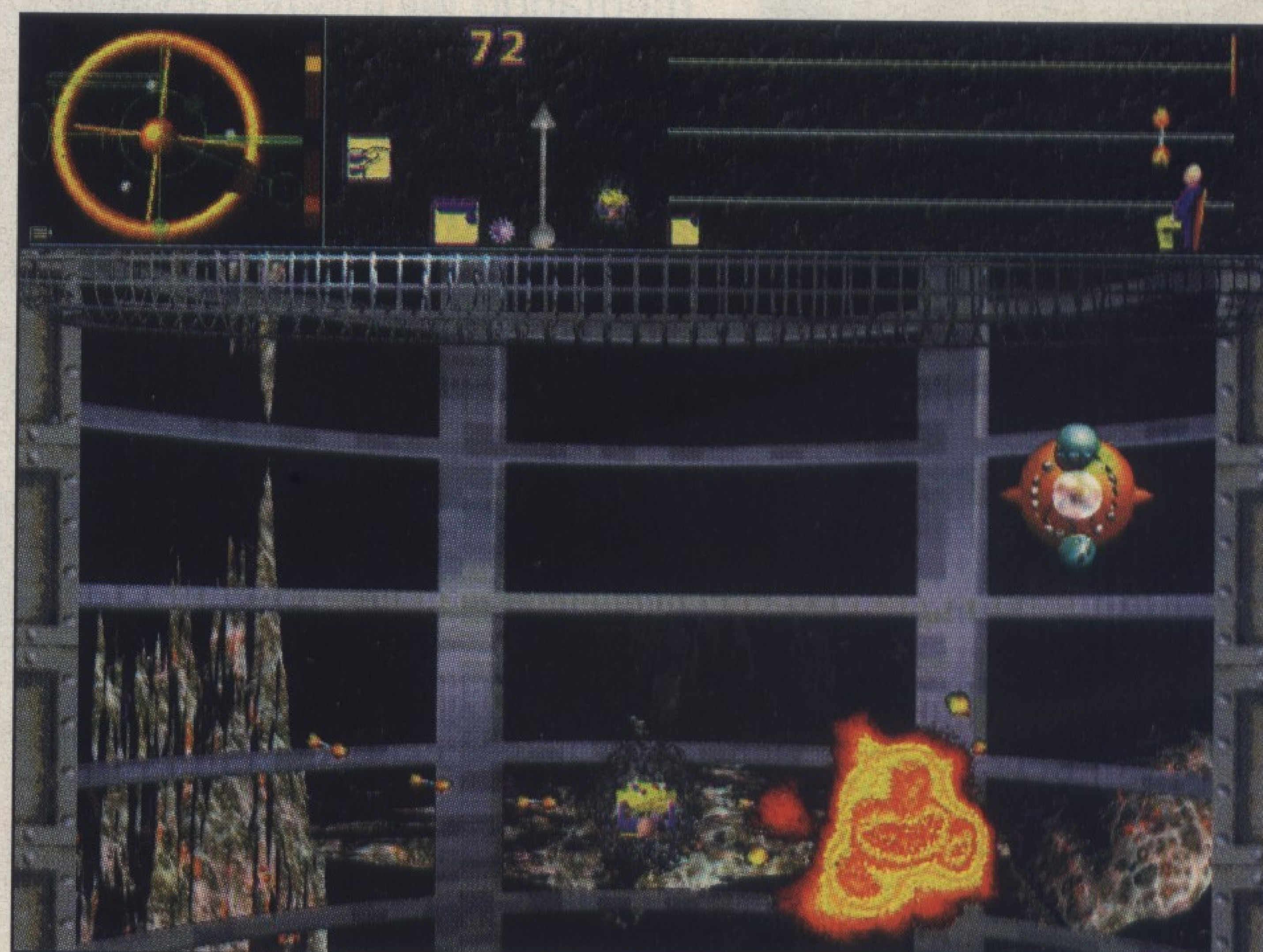
Men in Green

Un arcade al más puro estilo *quake*. Un programa de gran











calidad al que merece la pena echar un vistazo.

Numbers
La rebelión de los números. Cansados de figurar en el papel, los números han saltado a las plataformas y se dedican a machacar todo lo que se cruza en su camino.



CURSOS Y EJEMPLOS

Las líneas de código de los cursos de la revista, completos y listos para que los uséis. Pequeños programas que os ayudarán a mejorar y profundizar vuestros conocimientos de programación.



SHAREWARE

Y como siempre, una selección de las utilidades share que más os pueden interesar.

ACDSee 2.41
Uno de los más útiles visores en su última versión.



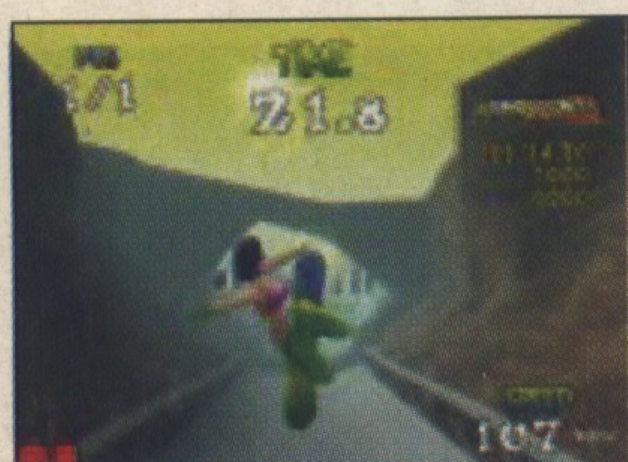
CD Box Labeler 1.1
Te permite crear etiquetas y libretos para CDs.

Copernic 99 301
Este agente de software gratuito encontrará con exactitud lo que estés buscando en Internet.

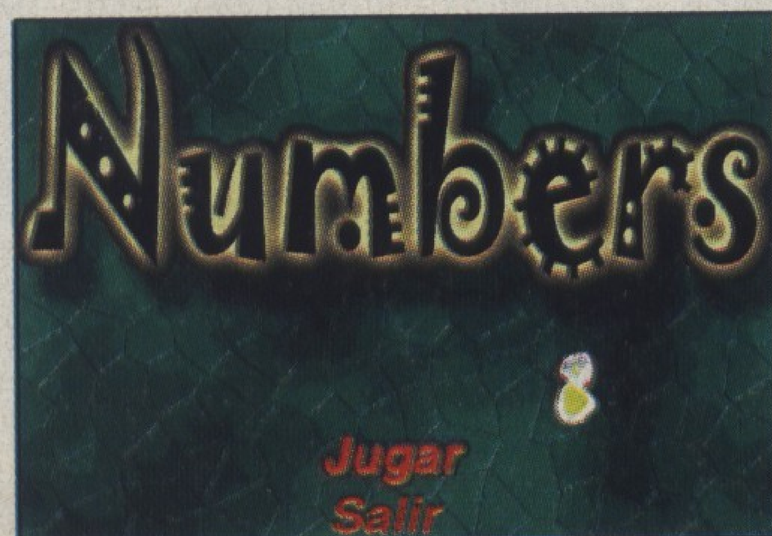


Cyber-Info Webmail
Hotmail Notify es un sencillo programa que te avisa cuando tu cuenta de HotMail tiene nuevos mensajes.

Download Accelerator 3.5
Download Accelerator acelera la recepción de ficheros fragmentando los mismos a partir de un tamaño definido por nosotros mismos.



Eudora Pro 4.2
El cliente de correo electrónico más popular para Windows 95/NT.



FTP SERV-U 2.5ª Build 4
Un excelente lector y buscador dentro de newsgroups. A la hora de buscar podrás especificar intereses diferentes, cada uno con su propio criterio, y escogiendo los newsgroups donde buscar.

Hyper Maker
Aplicación que te permitirá transformar páginas Web en publicaciones comprimidas y encriptadas, para su distribución. Incluye un visualizador libre de royalties.

mIRC 2.01
Es uno de los mejores clientes de IRC, además de ser todo un clásico. Si quieres charlar con gente de todo el mundo a través de Internet, no busques más.

Nendo 1.1
Nendo es una potente herramienta de modelaje en 3D que, sin embargo, destaca por su facilidad de uso. Utilizando su técnica de "arcilla digital", Nendo está basado en la escultura tradicional.

NetInfo 3.4
Programa multitarea totalmente configurable que proporciona herramientas para recopilar diversa información sobre la conexión actual de un ordenador a Internet.

News Rover 4.42
Útil lector de news, gracias al cual podremos organizarlas de forma cómoda y acceder a ellas sin ningún tipo de problema.

Organizador de Música
Esta excelente aplicación pondrá orden donde antes sólo había caos. Gracias a ella podrás ordenar perfectamente las bandas de música si tienes tu aparato conectado al equipo, así como las melodías que poseas en el estándar MP3.

PACT JumpReg 99.6
JumpReg te puede llevar hasta las palabras clave de más uso del Registro, ahorrando un montón de tiempo y esfuerzo.

PPPshar Pro 1.51
Con *PPPshar,* podrás conectar todos los ordenadores de tu red a Internet usando una sóla conexión telefónica.

SmartFTP 1.0 Build
SmartFTP nos proporciona una interfaz de múltiples ventanas con soporte para arrastrar y soltar.

SynEdit 0.40 beta
SynEdit es un excelente editor de textos orientado a programadores.

UltraEdit Professional
Es más que un reemplazo del Bloc de Notas, es un editor de textos completísimo. Soporta múltiples ficheros al mismo tiempo y de tamaño ilimitado.

URLSenty 1.14
Escanea de forma instantánea tus páginas Web favoritas en busca de cambios o actualizaciones.

Viruscan 4.0.3
Uno de los más importantes antivirus del mercado nos llega en su última versión shareware. Se trata de un programa imprescindible para tener nuestro equipo libre de virus.

WebAurora 98 1.01
El editor Web para Windows 95 y 98. Incorpora cinco asistentes para la creación de documentos y cuatro editores.

WinAmp 2.22
Gracias a varias características que lo hacen único en su género, con este programa podremos escuchar música con la más alta fidelidad.

Winzip70SR-1
El más importante compresor del mercado. Gracias a él podremos descomprimir gran cantidad de programas de juegos, incluidas algunas de las demos que os ofrecemos. Una herramienta imprescindible.

WorkTime 1.1
Con *Worktime* puedes controlar cuánto tiempo pasas delante del ordenador trabajando, ya que te avisa cuando tu jornada ha terminado. Cuenta con dos tipos de alarma (normal y de cuenta atrás), con soporte multiusuario y con un excelente generador de estadísticas.

CONTENIDO DEL CD ROM



ANCIENT EVIL
Una exclusiva. Os ofrecemos un juego de rol medieval en el que tendremos que luchar contra numerosos enemigos para desvelar el misterio de La Cripta de los Antiguos. Armas, hechizos, objetos mágicos, tesoros... todo un desafío.

WINTER SPORTS
Toda la emoción de los juegos de invierno es lo que nos ofrece este nuevo producto. En esta segunda parte del fabuloso *Snow Wave* tendremos la oportunidad de practicar 3 deportes de invierno apasionantes, el esquí, el snow, y el snowmobiling, o carreras de motos de nieve.

SEVEN YEARS WAR
Un curioso juego de estrategia con un planteamiento original. Como la mayoría de los juegos del género se sitúa en la edad media; sin embargo, la acción se traslada a las lejanas tierras de Oriente. Un buen programa para los amantes de los juegos de estrategia que sin duda encontrarán la manera de disfrutar con las aventuras que propone.

DIV GAMES STUDIO 2
Para los que se acerquen por primera vez a nuestra revista y que no hayan tenido la oportunidad de ver las maravillas de *DIV 2*.

REVISTA DIVNET
Para todos aquellos que no estén enganchados a la red pero tienen curiosidad por saber que es lo que se cuece en esta revista electrónica la hemos incluido en el CD. Quién sabe si de esta forma no os convenceremos para dar el paso definitivo y conectaros por fin a Internet.

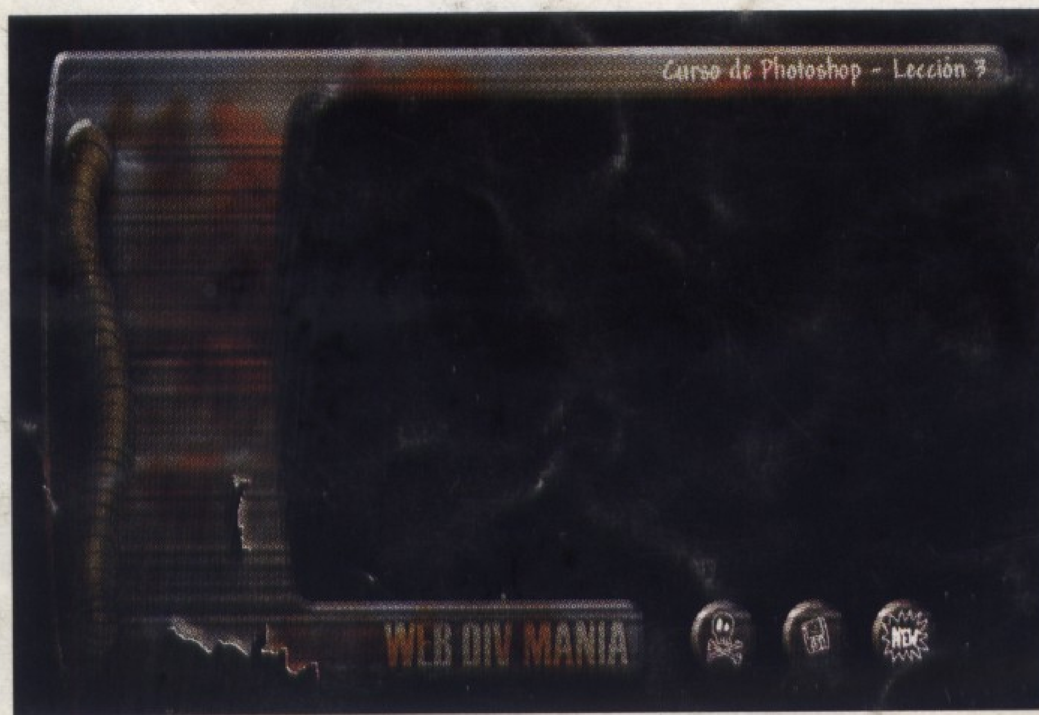
CURSOS Y EJEMPLOS
Las líneas de código de los cursos de la revista, completos y listos para que los uséis. Pequeños programas que os ayudarán a mejorar y profundizar vuestros conocimientos de programación.

SHAREWARE
Y como siempre, una selección de las utilidades share que más os pueden interesar.

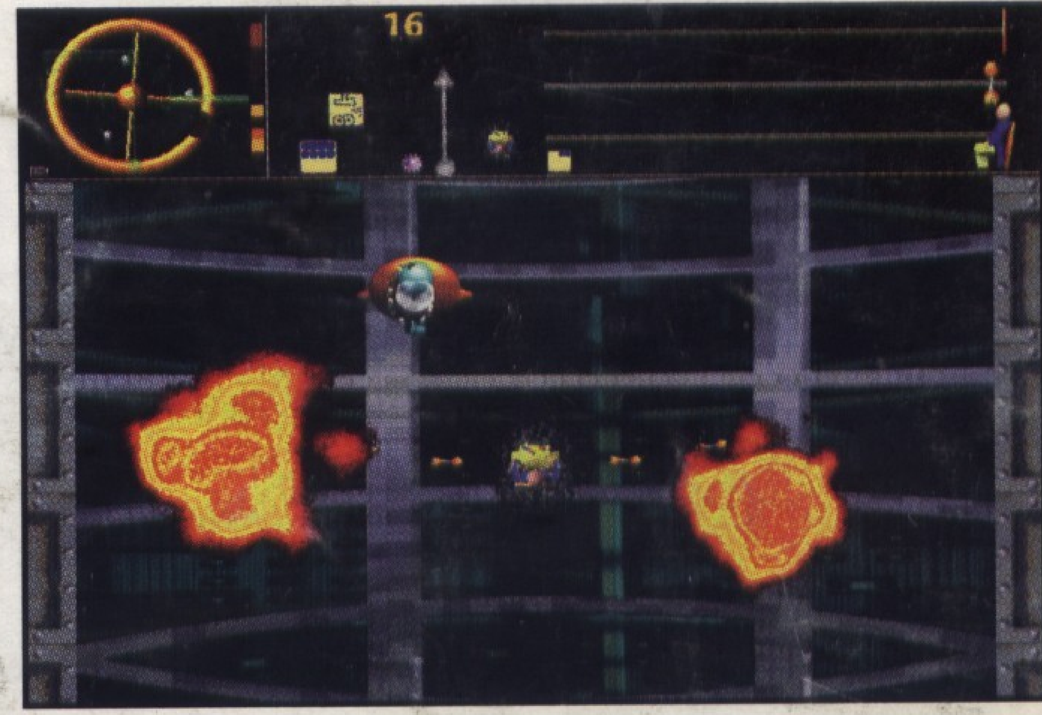
REPORTAJE. Los 10 mandamientos del programador independiente.



LIBRERIAS. Ficheros que os serán útiles para realizar vuestros programas.



PROGRAMA DEL LECTOR. Y los juegos ganadores de este número.





CON EL MEJOR CONTENIDO


Los Diez Mandamientos del programador independiente
O cómo programar todo un éxito


Cómo crear un videojuego
Lo que debes hacer para programar tus juegos


Introducción a la inteligencia artificial
Cómo programar un juego de rol

ACTUAL **EXHAUSTIVO** **DIDÁCTICO**

Y MUCHO MÁS...